AD-A273 362

②

Naval Research Laboratory
Washington, DC 20375-5320

NRL/FR/5546--93-9581

An International Survey of Industrial Applications of Formal Methods

Volume 1—Purpose, Approach, Analysis, and Conclusions

DAN H. CRAIGEN
ORA Canada

SUSAN L. GERHART
Applied Formal Methods

THEODORE J. RALSTON
Ralston Research Associates

DTIC
[illegible]
DEC 0 3 1993

September 30, 1993

93-29528

93 12 3 074

Approved for public release; distribution unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave Blank)*

2. REPORT DATE
September 30, 1993

3. REPORT TYPE AND DATES COVERED
Final Report

4. TITLE AND SUBTITLE
An International Survey of Industrial Applications of Formal Methods
Volume 1—Purpose, Approach, Analysis, and Conclusions

5. FUNDING NUMBERS
PE - 63794N

6. AUTHOR(S)
Dan H. Craigen,* Susan L. Gerhart,† and Theodore J. Ralston‡

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Research Laboratory
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER
NRL/FR/5546--93-9581

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research
Code 01111
Arlington, VA 22202

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
*ORA Canada, Ottawa, Ontario
†Applied Formal Methods, Austin, Texas
‡Ralston Research Associates, Tacoma, Washington

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Formal methods are mathematically based techniques, often supported by reasoning tools, that can offer a rigorous and effective way to model, design, and analyze computer systems. The purpose of this study is to evaluate international industrial experience in using formal methods. The cases selected are, we believe, representative of industrial-grade projects and span a variety of application domains. The study had three main objectives:

- to better inform deliberations within industry and government on standards and regulations;
- to provide an authoritative record on the practical experience of formal methods to date; and
- to suggest areas where future research and technology development are needed.

This is the first volume of a two-volume final report on an international survey of industrial applications of formal methods. This volume describes the study, the formal methods, the cases that were studied, our approach to performing the study, and our analysis, findings, and conclusions. Volume 2 details the 12 case studies.

14. SUBJECT TERMS

Formal methods	Reasoning tools	System design
Computer systems	Formal specification	Design verification

15. NUMBER OF PAGES
106

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
UNCLASSIFIED

18. SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. SECURITY CLASSIFICATION OF ABSTRACT
UNCLASSIFIED

20. LIMITATION OF ABSTRACT
UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std 239-18
298-102

CONTENTS

EXECUTIVE SUMMARY E-1

1. INTRODUCTION 1

2. FORMAL METHODS 1

  2.1 A Historical Perspective 2
  2.2 What are Formal Methods? 3
  2.3 What are the Limits of Formal Methods? 4
  2.4 Specific Formal Methods 4

3. CASE SUMMARY 5

  3.1 Regulatory Cluster 6
  3.2 Commercial Cluster 8
  3.3 Exploratory Cluster 9

4. METHODOLOGY 9

  4.1 Areas of Interest 10
  4.2 Acquisition of Information 11
  4.3 Questionnaires 11
  4.4 Analytic Framework 12
  4.5 Cluster Analysis 14

5. REGULATORY CLUSTER 14

  5.1 Introduction 14
  5.2 Cases 16
  5.3 Observations 16
  5.4 Analysis 17

6. COMMERCIAL CLUSTER 21

  6.1 Introduction 21
  6.2 Cases 22
  6.3 Observations 22
  6.4 Analysis 23

7. EXPLORATORY CLUSTER 27

  7.1 Introduction 27
  7.2 Cases 27
  7.3 Observations 27
  7.4 Analysis 28

8. KEY EVENTS AND TIMING 32

8.1 Starter 34
8.2 Booster 35
8.3 Current State 35
8.4 Timing 36

9. ANALYSIS OF FORMAL METHODS R&D SUMMARY 36

9.1 Regulatory Cluster 36
9.2 Commercial Cluster 42
9.3 Overall Observations 46

10. FINDINGS, OBSERVATIONS, AND CONCLUSIONS 48

10.1 Maturing of Formal Methods 48
10.2 Scale of Application 49
10.3 Primary Uses of Formal Methods 50
10.4 System Certification 51
10.5 Tool Support 51
10.6 Technology Transfer 52
10.7 Formal Methods Skills are Building 53
10.8 Code Level Application of Formal Methods 53
10.9 Inadequate Cost Models 54

REFERENCES 54

APPENDIX A—BIOGRAPHIES OF AUTHORS 59

A1. Dan Craigen 59
A2. Susan L. Gerhart 59
A3. Theodore J. Ralston 60

APPENDIX B—FORMAL METHODS TECHNIQUES 61

B1. Software Cost Reduction (SCR) 61
B2. B 62
B3. Cleanroom 65
B4. Formal Development Methodology (FDM) 66
B5. Gypsy Verification Environment 68
B6. Hoare Logic 70
B7. Hewlett-Packard Specification Language (HP-SL) 73
B8. Occam/Communicating Sequential Processes (CSP) 75
B9. RAISE 78
B10. TCAS Methodology 80
B11. Z 82

APPENDIX C—INITIAL QUESTIONNAIRE 89

C1. Approach 89

APPENDIX D—QUESTIONNAIRE FOR STRUCTURED INTERVIEW 91

D.1 Organizational Context 91
D.2 Project Content and History 91
D.3 Application Goals 91
D.4 Formal Methods Factors 92
D.5 Formal Methods and Tools Usage 92
D.6 Formal Methods Tools Usage Questions 92
D.7 Results 93

APPENDIX E—REVIEW COMMITTEE 95

APPENDIX F—ACKNOWLEDGMENTS 97

Accesion For
NTIS CRA&I ☒
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1

DTIC QUALITY INSPECTED 3

EXECUTIVE SUMMARY

INTRODUCTION

Formal methods are mathematically based techniques, often supported by reasoning tools, that can offer a rigorous and effective way to model, design, and analyze computer systems. This report summarizes the results of an independent study of 12 cases in which formal methods were applied to the construction of industrial systems. A major conclusion of the study is that formal methods, while still immature in certain important respects, are beginning to be used seriously and successfully by industry to design and develop computer or computerized systems.

Canada's Atomic Energy Control Board (AECB), the U.S. National Institute of Science and Technology (NIST), and the U.S. Naval Research Laboratory (NRL) commissioned this study to determine the industrial track record of formal methods to date and to assess the efficacy of formal methods for meeting their needs. The study had three main objectives:

- to better inform deliberations within industry and government on standards and regulations;
- to provide an authoritative record on the practical experience of formal methods to date; and
- to suggest areas where further research and technology development are needed.

These objectives have been accomplished through the publication of this study. The study consists of two volumes and this Executive Summary. The first volume is the analysis of the supporting data contained in the second volume. Volume 1 includes a discussion on formal methods and a brief characterization of the formal and related methods used in the cases, a summary of the 12 cases, a description of the methodology used in the survey, a cluster-by-cluster analysis of the data, a discussion on the key events and timing associated with each case, an analysis of our formal methods R&D summary, and concludes with the findings, observations, and conclusions. The appendixes to Volume 1 contain brief biographies of the authors, brief characterizations of 11 formal methods used in the cases, the initial questionnaire, the questionnaire used in our structured interviews, and acknowledgments. Volume 2 contains detailed supporting data on the 12 cases.

Through these means, the sponsors have been provided with an organized body of systematic information, assessments, evaluations and observations that interpret and extrapolate useful data on cases of significant industrial scale and applicability to real-world products.

This Executive Summary presents:

- a summary of the major findings of the study.
- recommendations for continuing R&D in formal methods.

FINDINGS AND RECOMMENDATIONS

The following conclusions are the result of applying the expertise of the authors in analyzing the cases.

1. Formal methods are maturing, slowly but steadily. Ten years ago, formal methods were perceived as mainly proof-of-correctness techniques for program verification that were suitable only for use on toy problems. Today, we see two major differences. First, the general concept of formal methods embraces a wider perspective than proof-of-correctness to include formal specification, formal system design, formal design verification, and other steps in system engineering connoting the use of mathematical description and analysis of computer-controlled systems. Second, organizations responsible for a wide range of applications have found it necessary to reach out for some improved means of intellectual control over their complex system developments and have found ways that formal specification, design, and verification can be applied, along with many other techniques, to meet their needs.

2. Formal methods are being applied now to develop systems of significant scale and importance. The 12 cases studied were selected as representative of some 60 projects (completed and on-going) using formal methods on significant real-world systems. While a majority of these cases are European, there is an increasing number of North American examples. The 12 projects involve applications with highly complex requirements and multipage specifications, which cut across a number of modern application domains such as transaction processing, safety-critical control systems, embedded hardware, security, and VLSI design.

3. The primary uses of formal methods, as shown in the case studies, are re-engineering existing systems; stabilizing system requirements through precise descriptions and analyses; communication among various levels of system stakeholders (e.g., design team, QA, managers); and as evidence of "best practice" (in a regulatory and standards context).

4. Regulators are working with researchers and developers to develop practical and effective techniques for future system certification. Efforts such as the NIST FIPSPUB on high assurance software, D0178B, IEEE 1228, and the draft IEC/ISO regulations on product safety and software safety, to name a few, are emerging with requirements for improved design and development practices.

5. Tool support, while necessary for the full industrialization process, has been found neither necessary nor sufficient for the successful application of formal methods to date. Some of the 12 cases used tools and some did not. In general, the lack of tools did not impede the choice to use formal methods, since tools were developed or adapted as needed, but neither did the presence of a tool or tools stimulate the choice to use a particular method (except in the one case in which the use of a formal method was mandated).

6. Several North American organizations and many more European ones have formal methods technology transfer efforts in progress. Even those organizations which have seen successful applications show only a small degree of penetration in their projects on the whole and in degree of use within their software development processes.

7. Skills are building slowly within organizations that are attempting experimental formal methods use on industrial projects. The current educational base in the U.S. is weak in teaching formal methods in the context of software engineering.

8. Formal methods have been applied in a few instances at the code level of system development. However, most programming languages lack adequate semantic bases to support the full application of formal methods (as do many specification languages). Static analysis and compiler-type tools that have gained confidence may be employed to complement design

refinements carried close to a code level. However, many aspects of run-time environments and hardware, e.g., real-time and other performance aspects, remain to be treated adequately and must be viewed as holes in the general use of formal methods.

9. There are no generally accepted cost models to drive gathering data from current projects or predicting the paths of future projects. Some organizations, however, do have process descriptions and intuitive cost models in which they have sufficient confidence. It has been difficult to establish the value of formal methods in situations where already high quality can be otherwise achieved or where minimizing time-to-market dominates the product goals.

RECOMMENDATIONS ON R&D

From the authors' analysis of the 12 cases and the stated R&D needs from those we interviewed, the following areas are suggested for future R&D. Note that listing these areas is not meant to exclude other potentially fruitful research directions; these areas are drawn from the particular set of cases that we studied.

1. There is a clear need for improved integration of formal methods techniques with other software engineering practices.

2. While numerous formal methods-related tools exist, industry needs ruggedized versions; not research prototypes. These tools need to be an integral part of a broader software development tool suite.

3. Added emphasis on developing notations more suitable to use by individuals not expert in formal methods or mathematical logic is required.

4. Formal methods need to evolve with other computing science trends, such as visualization, multimedia, object-oriented programming, and CASE.

5. At least for those cases requiring regulatory approval, improved automated deduction support is required.

6. Expansion of formal methods capabilities to include real-time, concurrency, and asynchronous processes is needed.

7. Efforts at easing the transition of a complex technology, such as formal methods, to a broader user base is needed.

AN INTERNATIONAL SURVEY OF INDUSTRIAL APPLICATIONS OF FORMAL METHODS

VOLUME 1
PURPOSE, APPROACH, ANALYSIS, AND CONCLUSIONS

1. INTRODUCTION

Formal methods are mathematically based techniques, often supported by reasoning tools, that can offer a rigorous and effective way to model, design, and analyze computer systems. The purpose of this study is to evaluate international industrial experience in using formal methods. The cases selected are, we believe, representative of industrial-grade projects and span a variety of application domains. The study had three main objectives:

- to better inform deliberations within industry and government on standards and regulations;
- to provide an authoritative record on the practical experience of formal methods to date; and
- to suggest areas where future research and technology development are needed.

This study was undertaken by three experts in formal methods and software engineering: Dan Craigen of ORA Canada, Susan Gerhart of Applied Formal Methods, and Ted Ralston of Ralston Research Associates. Robin Bloomfield of Adelard was involved with the Darlington Nuclear Generating Station Shutdown System case. Brief biographies of the authors are included in Appendix A.

Support for this study was provided by organizations in Canada and the United States. The Atomic Energy Control Board of Canada (AECB) provided support for Dan Craigen and for the technical editing provided by Karen Summerskill. The U.S. Naval Research Laboratory (NRL), Washington, D.C., provided support for all three authors. The U.S. National Institute of Standards and Technology (NIST) provided support for Ted Ralston.

This final report consists of two volumes. This first volume describes the study, formal methods, the cases that were studied, our approach to performing the study, and our analysis, findings, and conclusions.

The second volume of the final report provides the details on the case studies. For each of the case studies, we present a case description, summarize the information obtained (from interviews and the literature), provide an evaluation of the case, highlight R&D issues pertaining to formal methods, and provide some conclusions. Earlier drafts of the case studies were reviewed by the relevant participants.

2. FORMAL METHODS

Before proceeding, we provide a historical perspective, explain the term "formal methods," and introduce the broad spectrum of formal methods techniques that are represented in the survey.

Manuscript approved April 14, 1993.

2.1 A Historical Perspective

For over two decades, researchers have explored topics in the mathematics of program synthesis and analysis. The article "Assigning Meaning to Programs" [Flo68] stated the goal of formalizing both semantics of programming languages, and specification and reasoning about individual programs. Attempting to achieve the goal led to the key idea of inductive assertions. Inductive assertions evolved into the use of relationships among preconditions, program statements, and postconditions for defining both language semantics and program meaning. The intriguing possibility of mechanical proof of programs, or alternatively, heuristic generation of programs, yielded many exploratory systems and theoretical insights. Two barriers to practical application arose: (1) it was difficult to capture the full semantic content of programming languages and operating environments; and (2) it was a constant challenge to express the functional and nonfunctional intent for a program in its context of use.

Research lead to development of many important concepts: formal definitions of complex language features and identification of pitfalls of unnecessary and overly complex features; specification languages for abstract data types, concurrent processes, and abstract machines; a theory of abstraction behind hierarchical system structures; mechanizable logics that permitted computational reasoning about program properties; and theories of domains such as security, synchronous clocks, microprocessors, and compiling. Practical applications were found in these domains and small-to-medium scale examples were worked out. Industrialization began in the U.S. about a decade ago through the government mandate of certification of security properties.

Practice went a different route. Verification was achieved (and defined) through case-based reasoning (i.e., testing) with numerous criteria and strategies for good testing practice (primarily functional and structural coverage). Reviews provided the primary means of intellectual control: mental checking of desirable properties of systems under development and the *concomitant* communication among stakeholders (managers, designers, testers, documenters, etc.). Heuristic methodologies for structured requirements analysis and design, which captured the conventional wisdom of good structure and provided a common means of communication, offered additional guidance toward systems.

Researchers developed a theoretical base for testing, and the results, although mostly negative, suggested various heuristics for testing that more closely approximated an ideal where each test case stood some chance of revealing errors or demonstrating new evidence of correctness. Heuristic methodologies from practice never gained much research attention although abstract data types gave rise to object-oriented languages and methods to add even more structure and support to heuristic system development. Theoretical results have also played a role in system development (e.g., data compression, error correction, and encryption algorithms for disk and network storage and data structures permit representation and searching of databases and processing of visual images). Especially demanding are theories and strategies for managing distributed computation and data on both physically distributed resources and multiprocessor computing systems.

No matter what technical approach is applied in software development, common information processing needs arise: maintaining consistency among, and intelligibility of, an interwoven mass of documents expressing the points of view of many stakeholders, with constant change in content and often change in structure of that mass, while the set of stakeholders also changes over what may be many years of a system's life. Programming environments have evolved to address this need: structured editors, configuration management, database representations, graphical interfaces, and ways of coordinating work flow among, as well as work products of, groups of system stakeholders. Particularly important are those assets that are viewed as worthy of use beyond their project context (e.g., software components, document templates, review guidebooks, error and productivity data).

Yet another thread in practice has been the greater attention forced onto the process aspects of system development: how an organization manages and improves its infrastructure and specific procedures. While the logic-based form of mathematical approaches to system description was maturing, so was another form: statistical reasoning about errors and growth of reliability over time, with the objective of introducing industrial quality control and assurance practices.

Thus we have the setting for this study and the present report: mathematical techniques have been maturing for 25 years while nonmathematical techniques and general concerns for process have driven the practice. In the past five years, sparse anecdotal evidence indicated that formal methods were beginning to be used in industrial practice, leading to sponsorship of the present study to determine systematically and factually where these applications were occurring, why, and how the methods were being used, and with what success.

2.2 What are Formal Methods?

Definitions of formal methods abound. In Craigen and Summerskill [FM89], formal methods are defined as:

> "Methods that add mathematical rigor to the development, analysis, and operation of computer systems and to applications based thereupon."
>
> "...are nothing but the application of applied mathematics—in this case, formal logic—to the design and analysis of software-intensive systems."

In more concrete terms, there has been a tendency on the part of the formal methods community to define formal methods in terms of a Hilbert-style axiomatization. For example, Robin Bloomfield [Blo92] has defined formal methods as follows:

> "A software specification and production method, based on a mathematical system, that comprises:
>
> - A collection of mathematical notations addressing the specification, design, and development phases of software production,
> - A well-founded logical inference system in which formal verification proofs and proofs of other properties can be formulated, and
> - A methodological framework within which software can be developed from the specification in a formally verifiable manner."

For the purposes of this report, we define formal methods as:

> *the application of mathematical synthesis and analysis techniques to the development of computer-controlled systems.*

In our view, the overriding reason for developing formal methods techniques is to erect a framework within which we can predict, in a scientific manner, the behavior of computer-controlled systems. While the techniques that underlie the cases that we have surveyed are stepping stones toward achieving such a scientific and engineering discipline, substantial work remains to be performed. The processes for developing computer-controlled systems are still evolving.

The reader should be aware that the terms "formal methods" and "program verification" are not synonyms. In our view, program verification is a part of formal methods where programs are proven to be consistent with their specifications. The style of such proofs is described by Gries [Gri81], Hoare [Hoa69] and Dijkstra [Dij76]; and supported in verification systems such as the Gypsy Verification Environment [GVE90]. We defer to Section 2.4 a more in-depth discussion of the formal methods techniques being used in our surveyed applications.

An NRL report [CGJ91] includes a glossary of terms that are often found in the literature on formal methods. While we have not explicitly attempted to conform to the definitions of terminology in that report, our usage does not vary substantially.

2.3 What are the Limits of Formal Methods?

As with any technology, there are limits to the applicability of formal methods. In addition to the practical limits arising from the current state of the technology, there are also theoretical limits. Understanding both kinds of limits is particularly important when formal methods are used in the development of critical systems.

We will defer to the conclusions any discussion of the practical limits of formal methods. Through the analysis of the cases, we studied how formal methods are being currently applied and may be developed. For example, it appears that the application of formal methods to real-time characteristics of systems is still premature. (Craigen and Summerskill [FM89] discuss theoretical and practical limits in some detail.)

The question of theoretical limits of formal methods has two aspects [FM89]:

- What are the boundaries between the real and mathematical worlds?
- What are the internal limits of mathematics?

To the first question, the boundary is the same as that of any applied mathematical domain: we can model physical processes, but we can never be sure that our model describes the actual workings of the physical processes. In addition, results in quantum mechanics guarantee that we are limited in what we are able to predict about the Universe. When we consider information (as opposed to physical) systems, we enter the realm of interpretation where more, or less, meaning may be conveyed than intended, the wrong meaning might be conveyed, or the wrong recipients might be involved. Often the requirements for such a system are influenced by human (rather than natural) laws, such as tax codes, risk factors, insurance policies, and professional standards.

Furthermore, with formal methods we must formally codify a client's requirements. This jump from the informal perspective of the client to a mathematical model can never be formally proven.

Since the early part of this century, there has been an increasing body of literature that has demonstrated limitations to our use of mathematics. Particular examples are the undecidability results (such as the halting problem).

2.4 Specific Formal Methods

The cases we investigated used a broad collection of formal methods. In Appendix B, we present summaries of the principal formal methods that are mentioned in the report. References for the various methods are included and our readers are directed to those references for in-depth presentations of the methods. Volume 2 of the Vienna Development Methodology symposium proceedings [VDM91] also includes tutorial presentations of a number of formal methods.

In Fig. 1, we associate the methods with the cases in which they have been used. The cases are summarized in Section 3.

- Software Cost Reduction (SCR): Darlington Nuclear Generating Station (DNGS)
- B: SACEM
- Cleanroom Software Methodology: COBOL Structuring Facility (COBOL/SF)
- Formal Development Methodology (FDM): Token-Based Access Control System (TBACS)
- Gypsy Verification Environment (GVE): Multinet Gateway System (MGS)
- Hoare Logic: SACEM
- Occam and Communicating Sequential Processes (CSP): INMOS Transputer
- RAISE: Large Correct Systems (LaCoS)
- Hewlett-Packard Specification Language
- TCAS Methodology: Traffic Alert and Collision Avoidance System
- Z: SSADM Toolset, Tektronix oscilloscopes, INMOS Transputer

Fig. 1 — Formal methods used in surveyed cases

In Appendix B, our summaries of the methods are divided into two parts: we discuss how the method works, and present some observations. We have organized our discussion on how the method works by identifying the

- representations used: What are the underlying notations?
- steps performed: How are the representations used?
- tools applied: What tools are generally used?
- roles involved: Who does what and what skill do they have?
- artifacts produced: What are the major products that are documented?

For our observations, we

- describe what the method achieves,
- describe limitations of the method,
- identify other techniques that are supported or required, and
- identify the user community.

3. CASE SUMMARY

Twelve projects were chosen as the object of our study. These projects can be divided into three clusters: regulatory, commercial, and exploratory. Regulatory cases exhibit safety- or security-critical attributes and, thereby, attract the attention of the standards communities and agencies, and the regulators who will license the product for use. Commercial cases are those cases that involve purely profit concerns. Finally, the exploratory cases are those cases where the organizations involved were investigating the potential utility of formal methods in their own settings.

The cases are international, involving organizations in Canada, the United Kingdom, the United States, and continental Europe. Available resources did not permit the inclusion of cases from Asia or Australia.

We believe that the cases collectively uncover many factors relevant to evaluating the efficacy, utility, and applicability of formal methods. The cases also demonstrate different uses of formal methods. Among these are

- "modelization," where formal languages (e.g., Z) are used to model systems,
- demonstrating conformance of code with specifications,
- demonstrating conformance of design with requirements, and
- the application of mathematical reasoning to solve difficult conceptual problems.

Finally, we believe that the cases encompass many of the anecdotal claims, both pro and con, regarding formal methods.

We summarize the 12 cases in the remainder of this section. The cases are introduced in the context of the clusters. Our analysis of the collection of cases will be based on these clusters. Throughout the report we will use abbreviations to identify the cases; these abbreviations are introduced with the name of the case. Figure 2 presents an idea of the size of the applications involved. Of course, "lines of code" (LOC) is a rather superficial measure and must be viewed with caution.

3.1 Regulatory Cluster

3.1.1 Darlington: Trip Computer Software (DNGS)

Ontario Hydro and Atomic Energy of Canada Limited (AECL) developed computer-controlled shutdown systems for the Darlington Nuclear Generating Station (DNGS). When difficulties arose in obtaining an operating license from the Atomic Energy Control Board of Canada (AECB), the Canadian regulator for nuclear generating stations, formal methods were applied to assure AECB that the software met requirements. The process was one of post-development mathematical analysis of requirements and code using Software Cost Reduction.

The specifications, code, and proofs require 25 three-inch binders for each of the two shutdown systems. While there is some discrepancy in the various papers on the amount of code for the two shutdown systems, the definitive word was that one of the shutdown systems (SDS1) has about 2500 lines of code.

The use of the Software Cost Reduction approach finally convinced the AECB that the shutdown system was no longer a licensing impediment.

3.1.2 Multinet Gateway System (MGS)

The Multinet Gateway System is an Internet device that provides a protocol-based datagram service for the secure delivery of datagrams between source and destination hosts. This case is our main computer-security application. (TBACS, one of the exploratory cases, also involved computer-security considerations.) MGS went through a significant portion of the U.S. Trusted Computer System Evaluation Criteria process [NCSC85] and achieved a "developmental evaluation." The process included TEMPEST and communications security analysis. Rigorous mathematics and the Gypsy Verification Environment were used to develop and model security functionality.

Regulatory	
DNGS (SDS1)	1362 LOC Fortran; 1185 LOC Assembler
MGS	10 pgs math; 80 pgs Gypsy; 6 KLOC OS
SACEM	9 KLOC verified code; total of 315,000 person hours
TCAS	7 KLOC of pseudocode; specs about the same size
Commercial	
SSADM	350 pgs Z/English; 550 schemas; 37 KLOC Objective C
CICS	268 KLOC new/modified code; 50 KLOC traced to Z specs
Cleanroom	(COBOL/SF) 80 KLOC; (20 KLOC reused; 18 KLOC changed; 34 KLOC new)
Tektronix	200 KLOC; 2 x 15 pgs of Z specs
INMOS	Floating Point Unit: 100 pgs Z; 4 KLOC Occam; Virtual Channel Processor about 10^6 states
Exploratory	
LaCoS	No scale reported
TBACS	300 lines of FDM; 2500 lines of C
HP	55 pgs HP-SL; 1290 lines of spec and design; 4390 LOC

Fig. 2 — Scale of applications

From the formal methods perspective, 10 pages were needed to describe the security model and a further 80 pages for presenting the Gypsy specification of the MGS. The underlying operating system has about 6,000 lines of code.

3.1.3 SACEM

The product developed is a certified safety-critical railway signalling system which reduced train separation from 2 minutes 30 seconds to 2 minutes, while maintaining safety requirements. The successful deployment of the signalling system removed the need to build a new third railway line. The developers of the signalling system were required to convince the RATP (the Paris rapid transit authority) that the system met safety requirements. Among the numerous techniques used to demonstrate system safety (e.g., fault analysis and simulation) were B and Hoare Logic.

The system consists of 9,000 lines of verified code and 120,000 hours of formal methods effort. The new system allows for 60,000 passengers to be carried per hour.

3.1.4 Traffic Alert and Collision Avoidance System (TCAS)

The purpose of the TCAS family of instruments is to reduce the risk of midair and near midair collisions between aircraft. TCAS functions separately from the ground-based air traffic control system. The U.S. Federal Aviation Administration has required, under Congressional mandate, that TCAS II be installed on all aircraft by December 31, 1993. The TCAS methodology was developed and used to formally describe two components of TCAS: the Collision Avoidance System (CAS) Logic and the surveillance system.

There were 7,000 lines of pseudocode to describe the CAS Logic; the formal description is of comparable size. Work on the surveillance system is in progress.

3.2 Commercial Cluster

3.2.1 Structured Systems Analysis and Design Method (SSADM) Toolset

The British firm Praxis developed this computer-assisted systems engineering toolset to support the use of the CCTA standard development method. Z was used to develop a 350-page formal specification of the toolset infrastructure, from which was developed 37,000 lines of Objective C code.

3.2.2 Customer Information Control System (CICS)

The Customer Information Control System is a large transaction processing system developed by IBM. A major portion of a recent release was re-engineered using the Z method and tools at IBM Hursley, U.K. CICS is approximately 800,000 lines of source code; of the approximately 50,000 lines of new or modified code, 37,000 were completely specified using Z and about 11,000 were partially specified using Z. IBM claims that the use of Z has resulted in reductions in both development cost and error rates.

3.2.3 Cleanroom Software Methodology

To better understand the Cleanroom methodology, we investigated two industrial applications: one at NASA Goddard and the second at the Federal Systems Division of IBM. The Goddard application focused on the attitude ground support for NASA's International Solar Terrestrial Physics Satellite. The second application, and the prime object of our study of Cleanroom, was the development of a COBOL Structuring Facility (COBOL/SF), which converted old COBOL programs to a semantically equivalent "structured programming" form. The COBOL/SF resulted in a product of 80,000 lines of code and which required 70 person-months of effort. The product was important for demonstrating to IBM management the potential of the Cleanroom methodology.

3.2.4 Software Architecture for Oscilloscopes Using Z (Tektronix)

Tektronix in Beaverton, Oregon, used Z to develop a reusable software architecture to be shared among a number of new oscilloscope products. Z was used as a mathematical modelling language to explore design ideas. The models were viewed as being "nonexecutable prototypes." The software architecture consists of 200 KLOC and 30 pages of Z.

3.2.5 INMOS Transputers (INMOS)

In 1985, a small group of designers at INMOS Ltd. in Bristol, England, began exploring the use of formal program specification, transformation, and proof techniques in designing microprocessors. INMOS manufactures advanced microprocessors, and their best known product is the Transputer family of 32-bit Very Large Scale Integration (VLSI) circuits with a unique architecture designed for concurrent multiprocessor applications (processor, memory, and communication channels are self-contained on each Transputer chip).

This case consists of three interrelated projects, all of which use formal methods in some aspect of the design or development of components of three generations of the INMOS Transputer:

1. The "floating point" project: the use of Z to specify the IEEE Floating Point Standard which was applied to two successive generations of Transputer (a software implementation, and a hardware implementation).

2. The use of Z and Occam to design a scheduler for the T-800 Transputer.
3. The use of Communicating Sequential Processes and Calculus of Communicating Systems plus a "refinement checker" in the design and verification of a new feature of the T9000 Transputer, the Virtual Channel Processor.

3.3 Exploratory Cluster

3.3.1 Large Correct Systems (LaCoS)

The Rigorous Approach to Industrial Software Engineering (RAISE) is a large language and toolset evolved from the Vienna Development Methodology (VDM). Following the funding of RAISE during an ESPRIT I (1985-1990) project and the commitment of commercialization by Computer Resources International (in Denmark), an ESPRIT II (1990-1994) project was formed with the dual purpose of (1) improving the industrial potential of RAISE and (2) transferring formal methods into various LaCoS partners. This survey interviewed one partner at some length with a brief interview with a second partner (there are a total of six "consumer" partners in addition to Computer Resources International, the "producer"). The first, Lloyd's Register, is evaluating RAISE (as well as other methods) on a data acquisition and equipment management system for ship engines. The second, Matra Transport, builds railway and other systems (like GEC Alsthom on SACEM) and has primarily been exploring methodology so far. Both partners are building up consultancy in formal methods and assessment. Since LaCoS is an ongoing technology transfer project on a large scale, this case will be important to follow for the remaining three years and beyond.

3.3.2 Token-based Access Control System (TBACS)

The National Institute for Standards and Technology (NIST) is the U.S. measurement and testing laboratory, with numerous ongoing projects underlying standards. In this case, one group in Computer Security Technology has been developing a series of prototype smartcards for cryptographic authentication for network access. Another group in Software Engineering is chartered to look at new technology in support of open systems and other commercial standards. In this case, a staff member from the Software Engineering group was interested in experimenting with formal methods and located the smartcard application as a basis. A toolset and approach was chosen that followed the standard process in Trusted System certification, using a theorem prover to verify that a design met requirements of a security policy model. In particular, TBACS is a smartcard access control system with cryptographic authentication.

3.3.3 Hewlett-Packard Medical Instruments

This case was of a significant scale and importance of product but it is considered exploratory in that the primary objective was technology transfer. The product is a real-time database (called the Analytical Information Base) of patient monitoring information. Using an HP-developed specification language (a variant of VDM), a specification was produced by a formal methods transfer group and a developer. The transfer effort failed because time-to-market was the key feature and formal methods offered little beyond the high quality already achievable by other means that were consistent with the culture of the organization.

4. METHODOLOGY

In the introduction, we stated that the purpose of this study of formal methods is to better inform industry and government, to provide an authoritative record, and to provide pointers to future needs. In this section, we discuss our criteria and priorities for information collection. We also discuss the process we used for acquiring information on the cases and our analysis approach.

As will be clear from the biographies of the authors (Appendix A), the members of the survey team are experts in formal methods and, to differing extents, have vested interests in the technology. Consequently, we must be concerned about potential bias in our analysis. One means of allaying this concern, and allowing for input from individuals who have expertise supplementing that of the study team, was to form a Review Committee (Appendix E). The Review Committee reviewed and commented on interim and draft final reports, and we have responded to their comments by making changes to the report.

The reader should note that our methodology was a "common sense" approach—as opposed to a specific social scientific approach—driven by the need for both general context of how methods were used and specific aspects of why formal methods were drawn and how they were used. In addition, many variables affect the success or failure of the case studies; consequently, we are limited in the purely scientific conclusions that can be reached.

4.1 Areas of Interest

As mirrored in the questionnaires (Section 4.3), we structured our areas of interest as follows:

a. *Characterization of organizations.* What were their motivations and how did the motivations relate to the principal goals of the various organizations? We also compare and contrast staff compositions of the organizations and project.

b. *Description of project.* How did the projects evolve and what levels of effort were associated with the projects?

c. *Description of project goals and motivations for use of formal methods (general) and choice of formal method (particular).* What forces were working on the project? What were the criteria for selecting the chosen formal methods?

d. *Description of the processes used in developing the application.* One of our motivations is our interest in the transition of formal methods into the organizations and the project. In general, how do organizations transfer new technology into their development groups? What are the educational requirements for the effective use of formal methods? How did the use of formal methods modify existing processes and how did the project's processes compare with general organizational application development processes?

e. *Description of the effects (both beneficial and detrimental) of using formal methods.* Specifically, we were interested in the effects formal methods had on achieving project goals.

f. *Description of the effects of tools.* What was the utility of the chosen (formal methods) tools and did the presence or absence of tools play a role in the choice of the formal method?

g. *Description of the qualitative and quantitative results of the project.* While interested in the general achievements of the project, we were primarily interested in the conclusions reached from using formal methods.

4.2 Acquisition of Information

On a case-by-case basis, we proceeded generally as follows:

1. An initial questionnaire was sent to the participants to be interviewed. In the majority of cases, the questionnaire was completed by and returned to the survey team prior to the interview. (The content and structure of the questionnaires are discussed in Section 4.3.)

2. The survey team studied relevant literature on the project.

3. Two members of the survey team interviewed project participants. A second questionnaire was used to structure the interviews. The members of the survey team shared the questioning and note taking. While the questionnaire usually structured the interviews, and thereby aided comparison across projects, unique attributes of projects led to some divergences in the interview process.

4. After the interview, the members of the survey team produced raw notes on the interview. These notes, along with information gathered from the literature, were used to supplement the initial and interview questionnaires. From the literature and interview notes, the study team wrote a report on each case, using a reasonably consistent framework. In particular, we developed an "analytic framework" (described below) for analyzing the cases. The reports were sent to project participants for comment.

4.3 Questionnaires

Two questionnaires were used in the survey. The first questionnaire was sent to the individuals being interviewed, and the completed questionnaire was often returned prior to our arrival at their site. The initial questionnaire had two main roles: to have the individual being interviewed consider his or her project in the light of our areas of interest; and to provide initial feedback to us on the project. The initial questionnaire is included in Appendix C. This questionnaire evolved from our experiences interviewing individuals at NIST (for TBACS) and at Carnegie-Mellon University (for Tektronix).

The second questionnaire includes questions of greater depth and was primarily meant to structure our interviews. Thus, we attempted to maintain consistency, covering the same material throughout all the interviews; we had reasonable success in achieving this consistency. The second questionnaire is included in Appendix D.

Both questionnaires were divided into six categories, with the questionnaires intended to complement each other. For example, the questions in the initial questionnaire are broader than the more technically focused questions of the second questionnaire. The six categories are as follows:

Organizational context. Basic organizational information was requested. Besides understanding the general composition of the organization, we attempted to compare and contrast the staff compositions of the organization and project. (We were intentionally vague as to what "organization" means. We expected our respondents to define what they felt was the environment that had the greatest impact on them.)

Project content and history. We obtained a description of the application, its evolution, and the resources used. We also obtained information in support of the comparison between the organizational and project compositions.

Application goals. We delved into the reasons why the application was developed and the major influences that led to or affected the development.

Formal methods factors. We were interested on the "why" and "what" of selecting formal methods.

Formal methods and tool usage. Having selected to use formal methods, what were the processes and tools that were used?

Results. Finally, we asked for general conclusions about the project and the impact of formal methods.

4.4 Analytic Framework

Through this study we obtained substantial information. By developing an "analytic framework," we set a context for analyzing the information, presenting our interpretations, and highlighting patterns.

For each case, we produced vectors pertaining to important features of the product being developed and to the process used in developing the product. Relative to the organization's usual approach to development (and this may be a subjective measurement), we characterized whether the use of formal methods played a positive (+), neutral (0), or negative (-) role. In our view, the limited time available for our study and the consequent level of detail being obtained from the cases did not permit a finer granularity of analysis.

There were instances where no information on a product feature or process characteristic was obtainable and, consequently, we were unable to submit any entry into the matrix. In such instances, we used "n/a" to identify our inability to draw a conclusion. However, some of the n/a's were a result of particular product features or process characteristics not being of relevance to a particular development. In the matrix we do not distinguish "not applicable" from "not available".

The product features and process activities included as part of our vectors are described below. The features and activities are not necessarily independent of each other. For example, client satisfaction will depend upon the cost and quality of the product. Keep in mind that we are attempting to measure the relative effect of using formal methods.

For each project, we provide an overview case description that includes the size of the effort, the stages of development, and project references; provide a profile of the interviews and a survey of the information obtained for the interviews and the references; and evaluate the case.

4.4.1 Background

Our background information includes the following:

Product:
What was developed? Who uses it?

Industrial Process:
What kind of development was performed? Briefly, what was the context?

Scale:
How large was the product development?
How extensive was the formal methods usage?

Formal Methods Process:
How were formal methods used? What tools were applied?

Major Results:
What were the highlights of the process and product features?

Interview Profile:
How much effort was expended, and by whom? What parties were interviewed?

4.4.2 Product Features

We measured the following product features:

Client satisfaction
Were clients happier with the product? (Happiness is, of course, due to many aspects of the product, including enhanced reliability and reduced cost.) Obtaining a baseline for satisfaction comparison can be difficult.

Cost
Was the overall cost of the product reduced, or the profit increased?

Impact of product
What was the effect of the production of the product on the organization? For example, was the product important to the company's profit margin or an organization's reputation?

Quality
Was the quality of the product improved? By "quality" we include concerns pertaining to safety and security properties, enhanced functionality and performance, and reduction in errors.

Time-to-market
Was the product, or family of products, made available for marketing more rapidly (or, at least, not further delayed)?

4.4.3 Process Effects

The process effects were divided into two parts: general and specific process effects.

General process effects:

Cost
This measure of cost is based upon reduction in effort.

Impact of process
What effect did the process used to develop the product have on the organization? For example, was the process important to the company's profit margin or an organization's reputation?

Pedagogical
As a learning opportunity, what did the organization make of the opportunity? Was there a steep learning curve?

Tools

Did the formal methods tools help or hinder the development of the product? Were the tools reliable?

Specific process effects:

Design

Were the designs produced using formal methods fundamentally better or worse in some respect? For example, were the designs simpler?

Reusable components

Did the use of formal methods ease the development and/or use of reusable hardware or software components, designs, abstractions, etc.?

Maintainability

Is it easier to maintain the product? Maintenance includes incremental updates to the product and fixing errors. Again, baseline data for comparison are difficult to obtain.

Requirements capture

Was the acquisition of requirements simplified?

Verification and Validation

What impact was there on the Verification and Validation aspects of the process? Observe that Verification and Validation includes testing, proofs, and reviews.

4.4.4 Observations

In this part of our evaluation, we highlight certain points about the case that we feel should be brought to the attention of the reader. We may, for example, include our opinions or information from other sources.

4.4.5 Formal Methods R&D Summary

Finally, we provide feedback to the formal methods R&D community by describing the methods and tools used on the project, and by providing recommendations based on the case.

4.5 Cluster Analysis

As described in Section 3, we divided the cases into three clusters: regulatory, commercial, and exploratory. Defining these clusters allowed us to analyze subsets of the cases that were reasonably comparable. For example, we found that there is increased concern for demonstrating that code is in conformance with requirements in the regulatory cluster than with the other two clusters.

5. REGULATORY CLUSTER

5.1 Introduction

While recognizing the substantial benefits that have accrued to society from the application of computer-based technology, there are risks involved with their introduction. Peter Neumann's Risks forum[1] is replete with stories of computers failing to perform as expected by their developers. The two failures that most caught public attention were the failure of the Therac-25 radiation therapy machine (resulting in at least three deaths) and the failure of the AT&T long distance network in 1991.[2] We will call systems whose failure may result in substantial damage to society (e.g., death, financial loss, release of toxic materials, unauthorized access to information) "critical systems."

A number of regulatory and standards organizations in North America and Europe have been investigating policy and technical approaches to achieving assurance for a range of critical systems. Formal and other rigorous methods have figured importantly in their investigations of the means that might be used or prescribed. Several standards, draft standards, regulations, and draft regulations covering a range of critical systems and adopting various approaches from mandatory use of methods to exhortatory provisions on "best practices" are currently circulating. These include the draft NIST Federal Information Processing Standards and special publications on high assurance software and cryptographic equipment, the Trusted Computer System Evaluation Criteria, IEC standards and draft standards, and the U.K. Ministry of Defence Interim Standard on Procurement of Safety Critical Systems (00-55, the draft European Commission Directive on Product Safety). It seems clear that no one standard or approach will satisfy the breadth of applications, and that a framework of integrated standards and/or regulations is evolving. The U.K. SafeIT Framework is one preliminary example.

The organizations charged with the responsibility for developing and applying these standards and regulations are concerned, therefore, with a set of specific issues that can be captured by the following questions:

1. What is the best policy approach to achieve assurance?
2. What should the project requirements be and how best can they be achieved reasonably?
3. How best to demonstrate conformance and to certify critical systems?

Therefore, with an eye to providing data and analysis of that data to help our sponsors (and industry) answer these questions, we chose four cases involving actual development of critical systems using formal methods.

It should be noted that regulators have different approaches to certification. We can distinguish at least three approaches:

1. certification of the product (e.g., TCAS by the U.S. Federal Aviation Administration),
2. certification of the process (e.g., AECB), and
3. certification of professionals (as proposed in 00-55).

---

[1] *Communications of the ACM* carries a monthly article by Neumann entitled "Inside Risks," which discusses topics related to computer-based risks. Neumann is also the moderator of the Internet Risks Forum, which is highlighted in Software Engineering Notes.

[2] See [Lee91] for an account of numerous computing failures, including the two mentioned here.

As described in introduction to Craigen and Summerskill [FM89], critical systems and formal methods may be viewed as being related as follows:

1. Critical systems are increasingly being controlled by computer systems.
2. Existing techniques for developing, assuring, and certifying computer-based critical systems are inadequate.
3. Formal methods have the potential for playing the same role in the development of computer-based systems as applied mathematics does for other engineering disciplines.
4. Formal methods have had limited impact on the development of computer-based systems and supporting technologies.

Interested readers should read the U.S. House of Representatives study [US89] for further discussions on regulatory concerns.

5.2 Cases

As described in Section 3, four of our cases were motivated by regulatory concerns:

- The Darlington Nuclear Generating Station (DNGS) Shutdown System - a safety-critical application.

- Multinet Gateway (MGS) - a security-critical Internet application.

- SACEM - a safety-critical railway signalling system.

- TCAS - a safety-critical midair collision avoidance system.

Volume 2 discusses the applications in more detail.

5.3 Observations

The use of formal methods in the regulatory cases appears to be motivated primarily by concerns of demonstrating high levels of assurance. While formal methods can be used proactively to develop better systems, there is also the need to be able to demonstrate to regulatory authorities, unambiguously and comprehensibly, the proposed effects of the systems (i.e., requirements) and to produce evidence (not only through test plans, but by proof) that the code conforms with the requirements. There appears to be a greater concern for proof in the regulatory cluster than with our other clusters. Hence, in the regulatory cluster, while modelization is clearly beneficial, there is the additional concern of definitively demonstrating a relationship between the model and the underlying code.

There thus appears to be a greater need for automated deduction support for regulatory projects. Not only is there a need for language processors, configuration management, tracing of requirements, etc., but there is a need for help with the proving process. Three of the cases had obvious needs for automated deduction support.

In the DNGS case, the use of formal methods was not the result of a "grass roots" movement, but was recommended by other organizations (AECB and their investigator, David Parnas). Similarly, for the Multinet Gateway project, the use of formal methods was mandated by contract, as it was required to achieve appropriate levels of assurance for its security functionality. On the other hand, the use of formal methods in SACEM and, perhaps to a lesser extent TCAS, was the result of interest on the part of the participants.

Given the importance of the products being produced, with respect to safety- and security-critical concerns, it is all the more important that the limits of formal methods, both in philosophic and in technical terms, be appreciated by regulators, developers, and the general public (see Section 2.3).

5.4 Analysis

Below, we discuss each of the features, and we attempt to draw some general principles. Figure 3 summarizes our evaluations.

5.4.1 Client Satisfaction

In all four cases, we found that the clients (generally viewed as the regulatory agencies) were satisfied with the formal methods efforts.

REGULATORY	DNGS	MGS	SACEM	TCAS
Client Satisfaction	+	+	+	+
Cost (of Product)	n/a	n/a	n/a	n/a
Impact	+	+	+	n/a
Quality	0	+	+	n/a
Time-to-Market	n/a	n/a	n/a	n/a
Cost (of Process)	-	n/a	0	n/a
Impact	0	0	+	+
Pedagogical	+	+	+	+
Tools	n/a	0	+	n/a
Design	+	+	+	+
Reuse	n/a	+	+	n/a
Maintenance	n/a	n/a	n/a	n/a
Requirements	+	+	+	+
V&V	+	+	+	n/a

DNGS = Darlington Nuclear Generating Station
MGS = Multinet Gateway System
SACEM = Railway Signalling System
TCAS = air Traffic Collision Avoidance System

+ = positive role
0 = neutral role
- = negative role
n/a = not available or not applicable

Fig. 3 — Evaluations of regulatory cluster cases

Pivotal to client satisfaction were the perceived gains in assurance and comprehension, for example, demonstration that code satisfied system requirements was fundamental to obtaining regulatory approval for SACEM and DNGS. In the DNGS case, the demonstration was crucial for removing the shutdown system as a "potential licensing impediment." With respect to comprehension, it was through the use of mathematical-based modelling that sharper understanding of system behavior was obtained. For example, all involved with the Multinet Gateway System found that their understanding of the security aspects of the MGS were improved.

Similar results are being achieved with the TCAS work. Our impressions are that the SC147[3] committee finds the formal description of the Collision Avoidance System Logic and the surveillance system to be substantially superior to the original pseudocode and English specifications. Reviewability has been substantially enhanced.

5.4.2 Cost of Product

While we were unable to obtain sufficient information to draw strong conclusions about the costs of the products being produced, a few observations are of interest.

It would appear that the level of effort directed at obtaining assurance and comprehension of the systems is proportionately higher in this cluster of cases than with the others. Substantial effort was particularly noticeable in the DNGS and SACEM cases. With DNGS, it was estimated that about $4 million was spent on the verification effort, it used up to 30 people at one point during the verification, and (arguably, especially since DNGS is having ongoing problems) it may have delayed licensing by two to three months. Each month's delay cost Ontario Hydro $20 million in interest payments. The expense of this effort is partly due to the immaturity of the technology that was being applied. In addition to applying the technology to the shutdown system, it was also necessary to refine and augment the technology. Tool support was effectively nonexistent and would have been, we believe, of substantial benefit.

One of the primary motivations for the SACEM project was to increase throughput and thereby eliminate the need for a third railway line (and the associated rolling stock, labor, etc.) costing hundreds of millions of dollars. While increasing throughput, it was, however, necessary to maintain safety margins. Some 310,000 hours were expended in developing the prototype and commercial systems.

With TCAS, the effort is one of capturing the requirements for an already deployed and, apparently, troublesome system. The costs involved with this project are upgrade costs associated with developing the formal description of the Collision Avoidance System Logic and the surveillance system, and the internal verification and validation effort.

5.4.3 Impact of Product

In general, we viewed the impact of the products as being positive for the organizations involved. Perhaps the clearest case is SACEM. Not only has the Paris transportation authority increased the throughput on one of its busiest lines, it avoided building a new line. In addition, GEC Alsthom was able to deliver the product and is successfully using its new-found expertise to market itself further.

Note that TCAS is already under deployment in the U.S. and that the formal methods exercise has yet to have any effect on the TCAS systems that are already flying; in effect, the case study is a reverse engineering exercise on the requirements for TCAS.

---

[3] An industry and government committee that is developing the TCAS requirements.

5.4.4 *Quality*

From the developer's perspective, formal methods helped to achieve quality goals in the MGS and SACEM cases. With respect to MGS, the mathematical analyses and modelling that were performed improved the understanding of the security principles. With SACEM, RER (the Paris regional rail system) quality requirements were met, even though increased functionality requirements were imposed (e.g., increased throughput). The DNGS is an interesting case study in that the use of formal methods improved assurance, but the regulators did not claim an increase in quality. In effect, formal methods were used *a posteriori* to model what existed and to demonstrate that the code met the requirements; formal methods were not used as part of the development process.

It should be noted that formal methods were not, and cannot be, the only techniques used to provide assurance. For example, SACEM used, among other techniques, graphical simulation, real-time simulation, Failure Mode Effects Analysis, Hazard Analysis, and Fault Tree Analysis.

One underlying theme, however, was that no one believed that testing alone could provide the necessary assurance; the anecdotal evidence arising from testing regimes was viewed as being insufficient.[4]

With respect to TCAS, while we cannot argue that TCAS itself has yet been improved, the quality of the review process has certainly been improved by replacing a large, difficult to understand, English language specification with a more precise and concise formal description that was readable by the TCAS SC147 Committee.

5.4.5 *Time-to-Market*

Time-to-market does not seem to have been a primary concern for the projects; though, especially noticeable with the DNGS project, delays can have a financial impact.

The possible effects of failing to complete product development or to achieve licensing can be extensive. For DNGS, Ontario Hydro would have had to revert to a hardware solution at a cost of about $1 million and a one-year wait for parts. For SACEM, a new railway line (costing hundreds of millions of dollars) would have been required.

5.4.6 *Cost of Process*

We did not obtain enough information from which to develop general conclusions about process cost. We observed, however, that substantial efforts were directed at attaining assurance and comprehension. The amount of effort appears to be substantially more than found in our other case clusters. The effort reported in the DNGS (four million dollars) and SACEM (approximately 120,000 hours) cases is substantially more than one would expect in future such projects. Both cases were using new and evolving technology; hence, efforts were needed to develop not only the product but also the technology being used.

---

[4] Butler and Finelli's paper "The Infeasibility of Experimental Quantification of Life-Critical Software Reliability" in the Proceedings of the ACM SIGSOFT'91 Conference on *Software for Critical Systems* discusses some of the limits of testing.

5.4.7 *Impact of Process*

The regulatory cluster of cases resulted in a split neutral/positive rating for impact of process. In the SACEM and TCAS cases, there are statements of intent to continue using formal methods in future projects. In fact, the developers of SACEM have already applied their techniques to two subsequent systems, Calcutta and KVS (see Volume 2 for description), and they intend to market their skills in developing other railway-based systems. They also intend to market a toolset. The neutral ratings for DNGS and MGS result from two different observations. For DNGS, it has been realized that for future developments of computer controlled safety-critical processes, formal methods must play an important role. On the other hand, the experiences that arose from DNGS have lessened the enthusiasm of some managers to shift from hardware controllers to software controllers. (This is one of the negative influences of using a technology that was immature and the consequent increase in cost.) For MGS, the impact is reduced by the contract-driven nature of the organization; novelty is not always viewed positively in responding to proposal requests. There was no clear continuation of formal methods projects at Loral and, in fact, an explicit decision (not related to formal methods) was made not to proceed with an attempt to acquire a high security level for the MGS.

5.4.8 *Pedagogical*

All organizations benefitted, educationally, from the projects. Better understanding of formal methods techniques, research and development issues, and the products were achieved. Also of importance was an increased understanding (especially in the DNGS and MGS cases) in the kinds of process necessary for regulatory requirements. Note that with the DNGS and TCAS there has been substantial feedback from the applications to the research programs of David Parnas [Par90] and Nancy Leveson [Lev91], respectively.

5.4.9 *Tools*

No general conclusions can be drawn from the use of tools in our regulatory cluster of cases. There appears to be some need for automated deduction support, especially for demonstrating consistency of code with specification. We note that there is an apparent dichotomy in that the MGS and SACEM projects made substantial use of tools, whereas, from a formal methods perspective, DNGS and TCAS used nothing. This dichotomy is, however, more in appearance than real since in both the DNGS and TCAS cases the lack of tools is due directly to the immaturity of the technologies being used.[5] Tool support was recognized as being needed in both cases; no philosophical positions against the use of tools were taken.

5.4.10 *Design*

Formal methods appear to have uniformly improved either the product or the understanding (and consequent assurance) of how the product works (in contrast to what was expected from conventional development methodologies). These improvements appear to be the result of

1. the mathematical analyses of domain theories (e.g., in MGS, exploration of security models),
2. mathematical arguments demonstrating properties of code and specifications (e.g., the code proofs, using B, of SACEM), and
3. the use of mathematical-based notations to describe requirements (e.g., TCAS) and specifications (e.g., DNGS).

---

[5] In TCAS, the commercially available STATEMATE did not meet the project needs, and a variant was developed.

5.4.11 Reusable Components

For the two cases where we could derive a conclusion, formal methods played a positive role in reuse. The security and safety models that were developed as part of the Multinet Gateway and SACEM projects, respectively, can be used in the development of similar systems. Note that we are claiming reuse with respect to "models" or "mathematical theories," not with respect to code. While we expect formal methods to be beneficial in the development of reusable code (because of the use of formal specifications to unambiguously describe the functionality of the code and the use of mathematics to develop sufficiently general code that is more likely to be appropriate for reuse), such benefits were not forthcoming in the four cases we studied here.

5.4.12 Maintenance

No general conclusions can be drawn from our regulatory cluster with respect to maintenance. We expect that formal methods will be beneficial for maintaining systems as long as they are a part of a well-defined and documented process. The use of formal notations to capture specifications and requirements (and properly using information hiding and modularization techniques) should increase comprehension and clarify interactions between parts of systems, thereby making maintenance simpler. However, if the benefits are to be ongoing, maintenance and product improvements must build upon and continue to use the practices that have been put in place.

5.4.13 Requirements

In all the regulatory cases, we were told that formal methods improved intellectual control, clarified requirements, and removed superfluous requirements. For example, the SC147 committee believes that the University of California at Irvine Statecharts-like notation has been beneficial, in that they are now arguing over the technical substance of the formal descriptions of the CAS Logic and surveillance system; they are not arguing over minor grammatical matters of a large English specification. Requirements have been clarified in the MGS case through a sharper understanding of the security principles at play in a heterogeneous network.

5.4.14 Verification and Validation

The use of formal methods was crucial to achieving assurance in the critical systems. Proofs that code was in agreement with specifications were of fundamental importance in the DNGS and SACEM cases. There was also some use of formal descriptions to generate test cases and the formalisms were used to complement other Verification and Validation procedures. The authors are strongly of the opinion that a spectrum of Verification and Validation technologies (e.g., formal methods, testing) must be used to attain adequate levels of assurance.

6. COMMERCIAL CLUSTER

6.1 Introduction

The cases examined all concern the use of a formal method to develop products that are sold (or, in the case of SSADM, were intended to be used) in the commercial marketplace. The companies performing the design and development of these products are primarily commercial companies (although each company has performed government contract work), which means they are subject to the "laws" of the commercial marketplace to stay in business. Much of the anecdotal reporting of formal methods experience in North America over the past decade has put forth the question "but what is industry really doing with these methods?" (as opposed to government) as the acid test of their efficacy. In a study such as this, therefore, it is important to have serious representative cases from the commercial sector, given the historical support governments in North America and Europe have given to the use of formal methods in security.

Therefore, the context in which we have examined these five cases focuses on commercial concerns that tend to influence the perceived success of a product in the marketplace: time-to-market, impact of the product, cost and cost-benefit, and quality.

It is noteworthy that for a number of these parameters, we were unable to make a definitive assessment for our impact vectors other than "n/a". A rating of "n/a" does not, however, mean no useful lessons could be drawn from the experience, and we have attempted to describe these lessons in the observations below. This inability occurs because sufficient data was unavailable on which to base a more definitive judgment, either because it was not collected or because it was proprietary. Much of the data that would typically be collected on a development project was not collected in these cases because typical metrics used in the industry either were not used, or in the few cases that tried (e.g., SSADM and HP), they were unable to produce reliable data because of the incompatibility between formal methods and more conventional approaches for which these metrics seemed more suited. In the one case that did attempt to collect comparative data (CICS), the developers claimed it was possible to compare the data collected on the formal methods-based parts of CICS/ESA 3 Release 1 with data collected for a previous release which did not use formal methods (from which comes the claim of a 9% reduction in total development cost).

6.2 Cases

- SSADM - a Computer-Assisted Systems Engineering tool
- CICS - a transaction processing system
- COBOL/SF - a restructuring program, using the Cleanroom methodology
- Tektronix Oscilloscope - a software architecture
- INMOS - a Floating Point Unit and Virtual Channel Processor

6.3 Observations

All the cases in this cluster show that formal specification methods can be used efficiently and productively. This distinction between "formal specification" and "formal verification" is made because four of the five cases avoided refinement as either unnecessary or impractical at this time. The experience of INMOS in using proofs for verifying properties of the design of the Virtual Channel Processor certainly shows some potential for refinement meeting efficiency, quality, and productivity goals; however, it is not possible to generalize from one case. Proof does not seem to carry the same level of importance to developers of commercial products as it did in our regulatory cases.

The lack of effective cost-benefit models and performance metrics hinders the analysis of the efficacy of formal methods in applications such as the ones in this cluster. The mismatch between techniques for measuring conventional approaches to software development and the formal methods process results in data that is either not comparable or does not do justice to the strengths and weaknesses of the formal approach.

It is difficult to discern a common reason or set of reasons (in terms of commercial, market-driven factors such as saving money, time, human resources, or meeting a delivery deadline) why the developers chose to use a formal method, except to note the general belief that it would provide more clarity to the specification. At least two of the cases (CICS and SSADM) related this clarity to "intellectual control" of the system, which can be traced to quality concerns.

Looking at these cases after they were completed, it is possible to see that there were some advantages gained beyond this clarity. In the SSADM case, Praxis discovered ways in which their Z specification and an object-oriented implementation complemented each other to achieve improved productivity over what would have been predicted. INMOS discovered they could save time by not performing as much coverage testing as they would have done without using the formal method. In the COBOL Structuring Facility (COBOL/SF), IBM realized a reduced maintenance burden. INMOS and Hewlett-Packard (in the exploratory cluster) found ways to use the formal specification to generate test cases more efficiently.

However, the formal methods were found to be unsuitable for several important problems. Both Tektronix and CICS found it difficult or impossible to specify interfaces using Z. Similarly, INMOS found that the real-time properties in the Floating Point Unit and Virtual Channel Processor could not be modelled adequately, and used conventional simulation techniques for these (which proved adequate).

Since four of the five cases in this cluster used Z in part or entirely as the specification language, one might be tempted to conclude there are aspects of Z that lend it to commercial applications. This may just be an anomaly, and that the choice of Z was dictated by circumstances peripheral to its features as a specification language, such as proximity of experts (for training and consultation), availability of tools (albeit primitive), strong salesmanship from Z advocates, a "push" from governmental organizations; or bias in our own selection of the cases. However, Z does have certain features that make it amenable to the sort of specifications illustrated by these cases. It is reasonably mature with respect to a set of conventions that might be called the Z style: a standard notation for sets, function, and logic; a multi-purpose structuring concept (the schema); a presentation style that permits mixing schemas with text; and a "social process" which stresses analyses in the form of reviews and oral explanations. The main effect of these conventions is to make Z reasonably readable.

6.4 Analysis

In this subsection, we discuss each of the features in turn. We attempt to derive some general principles in our discussions. Figure 4 summarizes our evaluations.

6.4.1 Client Satisfaction

In four of the five cases (SSADM, CICS, COBOL/SF, and Tektronix) we viewed the "client" primarily in terms of the end customer for the product, rather than the developers who used the formal method. In the INMOS Transputer case, we considered the microprocessor design and engineering team to be the customer.

We used a subjective measure of "satisfaction" which we generally characterized as "was the customer happy with the product." Recognizing "happy" can mean a multitude of things, we used the following subjective evidence to indicate at least some reason to be happy that came from the interviews: (1) a continued use of the product; (2) few or fewer (than a previous release) errors reported after delivery; (3) absence of legal action[6]; and (4) indirect evidence from marketing or sales organizations. Clearly, this is not a scientific measure. Yet it seems to us to have relevance in that the commercial computer industry tends to be driven by concerns about keeping the customer satisfied, and companies use this "measure" in the form of testimonials as a legitimate marketing feature. However, this particular measure cannot, in our view, be dominant when examining the experiential record for a particular method or technique, whether it be formal or not.

---

[6] This is not as facetious as it may sound given experiences such as the lawsuits over the Therac 25 microprocessor and VIPER, which was claimed to have been formally verified.

COMMERCIAL	SSADM,CASE	CICS	Cleanroom COBOL/SF	Tektronix	INMOS, Transputer
Client Satisfaction	n/a	n/a	n/a	+	+
Cost	n/a	n/a	n/a	n/a	+
Impact	n/a	+	n/a	+	n/a
Quality	+	+	+	+	+
Time-to-Market	n/a	n/a	n/a	+	n/a
Cost (of Process)	n/a	0	0	0	+
Impact	+	+	0	0	+
Pedagogical	+	+	+	+	+
Tools	-	+	0	0	+
Design	+	+	+	+	+
Reuse	n/a	n/a	n/a	+	+
Maintenance	n/a	+	+	n/a	n/a
Requirements	n/a	+	n/a	+	0
V&V	+	+	+	n/a	+

+ = positive role
0 = neutral role
- = not available or
n/a = not applicable

Fig. 4 — Evaluation of commercial case cluster

With the exception of the SSADM system, the customers were reported to be satisfied with the product after it was delivered and put into use. The client for the SSADM system was reported to be satisfied with the system as delivered but the product was not used for reasons not pertaining to formal methods. In the Tektronix case, the oscilloscope product has improved Tektronix market share and profits. CICS users have reported fewer errors (60% reduction in user reported errors requiring field repair), although the newer Z-based CICS releases have not yet been installed in most of the CICS sites. The formal methods group at INMOS has been elevated in the design/development process from a curiosity to a regular part of the critical design path.

6.4.2 Cost of Product

Unfortunately, we did not gather much data on the specific impact of formal methods on the cost of the products, other than the data from INMOS that the use of the formal method had saved them about 3 million Pounds Sterling and 3 months of coverage testing which kept the T-800 price competitive. Several of the interviewees noted that it was difficult to extract the impact of the formal method, other than to note that the time and resources in terms of personnel did not vary significantly from that which would have been allocated anyway. It would be easy to conclude intuitively that savings in the overall cost of development had been made through finding or preventing errors early and before they were discovered in testing or after delivery. However, without a better way to measure this claim, it is not possible to draw a conclusion one way or the other.

6.4.3 Impact of Product

In all five cases, the clients believed these products to be important to their lines of business. With CICS, the Tektronix oscilloscopes, and the INMOS Transputer, these products represent major revenue sources and are important to the reputation of the company. The COBOL/SF is on the market and is used, but we have no data on its impact on IBM revenues. For a majority of the cases, the developers considered the products as important or strategic. All were full scale, significant industrial applications on which continuing revenue and reputation depend. We believe, therefore, that it is fair to conclude that formal methods can be used on real-world commercial applications.

6.4.4 Quality

Most of the commercial cases had high quality requirements based on adding new features, and formal methods were chosen in part based on a prior belief that these methods would help meet these requirements by helping exert better control. For example, in CICS, the new release (Version 3) added significant new features and had to meet a high quality standard demanded by the marketing staff. Intellectual control of the development was also important in part to help meet quality requirements. In the Transputer case, reliability of a known but not understood component (the Floating Point Unit in the T414) and a completely novel component (the Virtual Channel Processor in the T-9000) were seen as critical to the overall quality of the product.

6.4.5 Time-to-Market

Insufficient information was obtained to draw conclusions about time-to-market. The timelines for some of these projects are of such duration that time-to-market does not seem to be a paramount concern (e.g., CICS, which is on an evolutionary path with regular biannual releases). In the Tektronix case, however, time-to-market was of paramount importance in that Tektronix was concerned about a stagnant market share with the potential for a decline, and Tektronix management was concerned about the risk of failing to release their new oscilloscope line in time by the use of formal methods. Management was persuaded that the reusable framework was worth waiting for, which was true as it has helped make the product successful in the market.

6.4.6 Cost of Process

This cost is weighed in terms of cost-effectiveness. It was impossible to assess this trade-off, chiefly because there are almost no means available to calibrate the formal methods process to fit the conventional metrics used by the industry. Eight of the twelve cases did not attempt to collect this data for this reason. CICS, COBOL/SF, and SSADM made attempts to collect this data, without much success. This argues that an important area for further work is in productivity metrics for formal methods that either fit conventional metrics or replace them.

6.4.7 Impact of Process

The impact of formal methods in the five commercial cases was almost uniformly positive. This reflects a positive impact in understanding and comprehension (CICS, INMOS, COBOL/SF), communication among team members (Tektronix, SSADM), and improvement to an existing or new process (CICS by introducing the Product Level Design stage in the Product Process Architecture; Tektronix by introducing more discipline and structure).

6.4.8 Pedagogical

All the companies benefitted educationally from the projects. There was better understanding of formal methods techniques, research and development issues, and of the products. There was also an increased understanding in the kinds of process steps necessary to meet the application goals.

6.4.9 Tools

Four of the five cases[7] used tools, and in two cases (CICS and INMOS) tools were specially developed to meet needs that evolved with the project. However, all the cases stressed the mental role of formal methods, and indicated that the tools helped on some simple mechanical tasks such as type checking and editing. In the interviews, all the developers indicated a desire for additional tools to help with some harder problems, such as refinement. Only INMOS built a tool to help check proofs for the Virtual Channel Processor but it was dictated by the task; none of the other projects did refinement but concentrated on formal specification. None of the projects indicated that tools either made or broke the effort. The conclusion, therefore, is that tools are nice to have when they are driven by the demands of the tasks but they are not necessary.

6.4.10 Design

The five commercial cases showed that formal methods had a positive impact on the system design in several ways. In the CICS and Tektronix cases, the use of Z as a language for modelling behavior at a high level of abstraction gave the developers intellectual control of the design: in CICS as an aid to documentation; and in Tektronix as a means of understanding the effects and other implications of requirements and communicating these systematically among the team members (which once understood gave way to code as the primary medium of communication). In the COBOL/SF case, formal reviews were used to evaluate quality factors in the design. Similarly, the reliability part of the Cleanroom method explicitly requires that very few errors should exist in the code when given to the independent test team, and the use of a formal modelling technique like box structures (or any other formal modelling language) helped achieve this goal.

6.4.11 Reusable Components

Only one commercial case, Tektronix, specifically set out to develop reusable components, and Z helped them achieve this goal. The flexibility of the Z notation helped express some difficult concepts, although it was not suited for others, such as the interfaces. In the INMOS and CICS cases, components such as the Floating Point Unit and several modules in CICS have turned out to be reusable in later versions, and the formalization played some role in this, although it must be noted that the reusability was an after-effect, not a primary requirement, and this result could have resulted from using other methods. Reuse was not a primary concern in the COBOL/SF case, even though parts of previous versions were reused.

6.4.12 Maintenance

The only case for which we collected any maintenance data was COBOL/SF, which has shown to need a small amount of maintenance (on the order of one person per year) which is substantially below the expected norms. The "process" of Cleanroom is perhaps more important to this outcome than the formal aspects per se, although the verification reviews seem to be an important contributor to the end result of fewer errors. The errors found and fixed were not deep design errors.

---

[7] COBOL/SF was the only exception.

6.4.13 Requirements

Requirements capture was aided in the CICS, INMOS, and Tektronix cases by using a formal specification language to express ideas that were either known but ambiguous (as with CICS) or novel (both INMOS and Tektronix) with greater precision than other languages. In the COBOL/SF case, the Cleanroom method claims to support requirements capture by forcing stabilization and pushing an incremental build approach. However, we did not discuss this claim in this application.

6.4.14 Verification and Validation

Each of the commercial cases showed a positive impact of the formal method on testing. In the main, this impact was achieved by using a formal specification to help generate test cases. (This also was the case in the Hewlett-Packard exploratory case and is the aspect of most interest to the Quality Assurance group at HP-Waltham.) INMOS also used formal specification and refinement (checking proofs) as an adjunct to testing for quality control purposes, and as a result, realized a significant time savings in coverage testing.

As an integral part of the Cleanroom approach, there is a relationship between the formal aspects of Cleanroom and the statistical testing through the verification reviews, but it is not clear how direct this relationship is. Indeed, there is a forced separation in the sense that the statistical testing step in the Cleanroom process is to be kept independent from the other steps in the process. However, there is also pressure from conventional practice to do some unit testing before sending the code to the statistical testing step.

7. EXPLORATORY CLUSTER

7.1 Introduction

The applications in this cluster of cases were primarily exploratory, although still well beyond the "toy problem" stage. Each application is based on an industrial-scale problem and formal methods were used, or were intended to be used, as part of the normal process of the organizations involved. These cases provide additional input to our analyses as well as providing a baseline against which to assess further uses within these organizations.

7.2 Cases

- LaCoS - primarily the Condition/Performance Monitoring Predictive System
- TBACS - a smartcard security application
- HP - an "analytical information base" for a patient monitoring system

7.3 Observations

We believe there are many other ongoing exploratory applications comparable to these (see [FM89] and [Sof91] for a few examples). In such cases, we would like to see gathering of data along the lines of this survey. We especially recommend using our background questionnaire and feature classification as a basis for either a progress assessment or a legacy.

Different technology transfer models are demonstrated in this cluster:

- LaCoS is a collaborative project specifically designed to achieve technology transfer, both of an individual toolset and of the general concepts.

- Hewlett-Packard was also a designed technology transfer experiment from a group with that charter to a willing individual in an application group.

- TBACS was a "grass roots" effort by an individual interested in learning more about formal methods and their tool support. He involved a second group that had a good application and exposed them to the ideas and possibilities of formal methods. This exploration increased the overall experience of NIST.

In all these models, there are interesting aspects of not only feedback regarding formal methods, but also of the cultural settings necessary for formal methods to take root.

The LaCoS project offered a luxuriously long opportunity for exploring and fostering interest in formal methods. In contrast, NIST and Hewlett-Packard afforded single individuals a brief time to experiment with formal methods along the lines of their own interests. That both NIST and HP acquired valuable data suggests that such experiments are worthwhile, but this limited resource commitment also can backfire if the experiment yields less than desired results (as may have occurred with both TBACS and HP).

In general, it is difficult to tell in exploratory cases whether success or failure was due to the formal methods used or the technology transfer process applied. Successful technology transfer depends upon the comitment of those involved (including management) and often takes substantial time and effort.

7.4 Analysis

In an exploratory case, the client and the actual product may be hard to define. In our analysis, we look at both the specific products that were subjects of the exploration and at the broader "product" of an experience base. Since this distinction is important, we focus on the individual features with respect to "process" where exploratory cases are easier to evaluate and group together all the product related features, looking at both the experience product and the actual products. Figure 5 summarizes our evaluations.

7.4.1 Product Features

- **The "Experience" Product:** All three organizations seemed to learn a great deal from their experiments, although project consequences are largely undetermined at the time of this study.

  LaCoS is a model for how organizations can acquire experience in formal methods and how a formal method product (RAISE) can be directed toward commercialization. The cost to the partners is lowered by ESPRIT support and the pedagogical costs are distributed among the partners. The ultimate effect on partners' business products is yet to be determined but the interviewed partners appear to be using LaCoS to build a consultancy presence. The LaCoS consumer partners seemed satisfied and were able to exploit the time offered (five years) to explore formal methods in general and the RAISE approach in particular.

  In TBACS, the technology transfer was disappointing. Despite a significant flaw and the evidence that their testing process was not adequate to find this flaw, other factors dominated the decision by the smartcard developers not to adopt formal methods. Regarding the exploration using the

EXPLORATORY	LaCoS	TBACS	HP
Client Satisfaction	+	n/a	0
Cost (of Product)	n/a	0	0
Impact	+	+	n/a
Quality	n/a	+	+
Time-to-Market	n/a	n/a	0
Cost of Process	0	0	0
Impact	0	0	+
Pedagogical	+	+	+
Tools	0	+	0
Design	+	+	+
Reuse	n/a	+	0
Maintenance	n/a	n/a	n/a
Requirements	+	+	+
V&V	n/a	+	+

+ =positive role
0 =neutral role
- =negative role
n/a =not available or not applicable

Fig. 5 — Evaluations of exploratory cluster cases

specific method and tool, it was found that there was a very good match and that a reasonable effort yielded a large outcome. The most important effect was to feed the experience into the creation of future standards.

At Hewlett-Packard, the experience was also two-sided, with one research/transfer group in England and the developer in the U.S. Strong efforts to train and support a user community failed: the HP culture was not ready to accept formal methods and the research/transfer group was unable to penetrate the culture via an unmitigated successful application of their technology. The formal methods products were usable and useful, and the encompassing Rigorous Software Engineering process generated suggestions for improving process for both the developer and the Quality Assurance group.

- **The Actual Products**: The LaCoS partners we interviewed were not yet applying formal methods to their company products. However, the consultancy businesses seem to be developing.

In the case of TBACS, the exploration did directly affect a product (the prototype) by finding a flaw in the smartcard, but we were unable to reach the end users of the smartcard. This was achieved with little added cost to the product because the exploration was funded by a different group. Estimates showed that the verification effort would have added little effort to the total development, therefore little cost to the product.

The product assessment is clearer in the Hewlett-Packard case. The database was not strategic, indeed software is not the dominant cost of the entire product. Therefore, the formal methods exploration did not add to the cost, but neither did it add to the quality, and more significantly, formal methods did not offer time-to-market gains.

7.4.2 Cost of Process

All three experiments with formal methods appear to be on a reasonable scale with assessments emerging soon after application.

In LaCoS, the learning process is deliberately planned for several years with considerable feedback. TBACS estimated that 6% of development cost would be attributed to the verification performed. Aside from start-up and training time, formal methods use did not alter HP development time, although the developer effort was less than the research/transfer group (which supplied "free" labor).

7.4.3 Impact of Process

An exploratory project may not yet have an identifiable effect on process. Two of these cases suffered discouraging responses in trying to affect existing processes.

With LaCoS, one significant impact of ESPRIT funding is that each partner has a group chartered to learn about formal methods, evaluate tools, and perform other duties. In TBACS, the process of using formal methods was not adopted directly for the project involved (TBACS) but could influence the organization in other ways (other standards). HP showed an interesting side-effect of formal methods usage by seeing ways to improve its process of review and traceability, although the specific formal method was rejected.

7.4.4 Pedagogical

An exploratory case reveals the quality of both the pedagogical material available and the ability of the organization to assimilate new knowledge and acquire new skills. Deeper investigation of these cases would help inform tool and method purveyors.

With LaCoS, pedagogical material is accumulating for RAISE and training is being transferred within the organization and to other partners. In TBACS, it was shown that a single individual could follow the security process using existing tools and tutorials when, as in this case, the match was good although the application was novel. The research/transfer group at HP learned how to carry their formal methods process through a complete development and the developer learned how formal methods worked on a problem.

7.4.5 Tools

An exploratory case may not have much time to incorporate tool use, but, on the other hand, tool experience may be the primary objective. Where advanced tools were used, the experience was favorable, but better tools would have helped in another case.

In LaCoS, a substantial toolset (language processors, configuration management, analyzers, and proof assistants) is recently available. However, the tools are not yet being used extensively for in-house partner experiments since learning the specification language is still the major objective.

The tools used in TBACS were "venerable" and did the job, but numerous deficiencies and needs were identified. At Hewlett-Packard, no major tools were used, but availability of the rudimentary set (editors, checkers) might have made an impression on developers.

7.4.6 Design

An exploratory case should carry a number of new techniques along with formal methods. These cases showed different kinds of interaction with the design process.

In LaCoS, a number of new concepts of design (e.g., ways of reaching code from specifications) are being taught and experimented with. TBACS was post-design but served to ask productive questions of the smartcard developer. The HP design showed that formal methods worked and illustrated new approaches not available previously.

7.4.7 Reuse

An exploratory case would view reuse, if not an explicit objective, as a fortuitous byproduct. These cases show different aspects of reuse.

LaCoS has the objectives of accumulating a library of specification components and of using the RAISE modularity capabilities, but nothing significant has been reported yet. TBACS exemplifies the reuse of models and modelling concepts, both by carry-over from past security models for other applications and as a stimulus for research in authentication logics and models. While no data was collected from HP about reuse, the product development was essentially re-engineering that relied on developer domain knowledge.

7.4.8 Maintenance

As maintenance was not an explicit objective for the exploratory cases, there were no significant results to report.

7.4.9 Requirements

Almost all cases, exploratory or not, have some requirements expression aspect. All three cases reported positive experience.

In LaCoS, there is considerable interest in modelization using RAISE as a way of capturing and expressing requirements for the type of systems they build. TBACS showed how a formal specification could be used to capture requirements and analyze a specification against them (potentially) early in the life cycle.

Formal methods were effective in communicating requirements between the HP developer and the research/transfer groups.

7.4.10 Verification and Validation

An exploratory case might or might not have a verification objective. In the one case where formal methods were used for Verification and Validation, there was a successful experience.

In LaCoS, there is interest in the RAISE method of proof justification but there was no activity in the partners surveyed. TBACS shows a standard "modelling with proof of putative theorems" approach for validation with no further investment in verification below the specification. The contrast in capability to testing is portrayed in the flaw that was found by proving and missed by testing. Although no verification was done with the HP database, the specifications were found useful in unit testing.

8. KEY EVENTS AND TIMING

In this section, we investigate key events associated with each of the cases. This particular analysis tries to determine patterns: "time span patterns" for how long it takes organizations to absorb or reject formal methods, and "booster patterns" for the people, events, or needs that played a key role in fostering and accelerating the case onward.

Consequently, our approach was to ask the following questions for each case:

Starter: What events or needs precipitated the use of formal methods in the case?

Booster: What events or needs led the project to continue to its current state?

Status: What is the current formal methods use with respect to the case and the organization?

For each case, we provide our perspective on the starter, booster, and current status. We follow the perspective with an analysis.

Project: SSADM Toolset (Praxis)

Starter: 1983: Formal methods used as in-house methods, consistent with quality/best methods philosophy of company.
Booster: 1987: First contract to include complete development from specification to implementation.
Status: Formal methods continue to be used on commercial and regulatory projects in conjunction with conventional methods.

Project: CICS (IBM)

Starter: 1981:Meeting of IBM manager (Tony Kenney) and C.A.R. Hoare, both looking for new ideas; established joint project for several years.
Booster: 1982-1985: Interaction of Hursley and Oxford during CICS system restructuring; need to gain control of documentation and system by re-engineering.
Status: Continuing use of Z for documentation and design; Research in Clean/Z project between Cleanroom and Hursley (combining aspects of Cleanroom with Z).

Project: Cleanroom (IBM)

Starter: 1982: Harlam Mills' view that a better software methodology could be put together from formal methods plus reliability testing plus reviews.
Booster: 1985: Increasing IBM concern about quality requirements plus maturing of Mills' ideas.
1988: Evidence from COBOL Structuring Facility application.
Status: Being explored in IBM, NASA, other companies trained by Mills, National Science Foundation workshop, book [Dye92], Cleanroom Technology Center transfer point in IBM.

Project: Darlington Nuclear Generator System (Ontario Hydro, AECL, AECB)

Starter: To 1989: Inability of Ontario Hydro and AECL to convince the AECB to approve the software used in the shutdown system.

Booster: 1989: David Parnas (consultant) recommended formalizing requirements, producing function tables for routines, and checking consistency rigorously.

Status: 1991: Shutdown system was approved and Darlington licensed. However, for the long term, the software needs to be re-implemented.

Project: LaCoS (Lloyd's Register; Matra Transport)

Starter: 1984-1988: Development of RAISE under ESPRIT funding as an industrial-strength (intended) formal method.

Booster: 1987: Acquisition of Dansk Datamation Center by Computer Resources International with intent to commercialize RAISE.
1991: Second round of ESPRIT funding with industrialization goal and ESPRIT/company subsidy.

Status: Large scale evaluation ongoing. Six partners learning RAISE (and other methods) to build up their business applications and determine value of RAISE.

Project: Multinet Gateway System (Ford Aerospace (now Loral))

Starter: 1981: Ford Aerospace moving into large-scale secure network business.

Booster: 1985: Contract that required the use of formal methods to achieve A1 certification.

Status: 1988: Achieved "developmental evaluation" but no further transition of formal methods technology into Loral at large.

Project: SACEM (GEC Alsthom, Matra Transport, Compagnie de Signaux et Entreprises Electriques)

Starter: Late 1970s: switch from hardware to digital software system control required new techniques; identification of Hoare axioms as possible proof technique.

Booster: 1984: RATP (the Paris transit authority) agreed that proof was acceptable and desirable (in conjunction with other techniques) and were trained to understand GEC's work.
1987: Review brought in Abrial and his B method. SACEM could have failed at this point in that (a) Hoare method could have continued and become too cumbersome, (b) B method might not have been adopted, or (c) the top-down/ bottom-up approach might not have worked.

Status: Full-scale industrial use in the railway switching domain. Projects use a defined method routinely by trained personnel with understood results. Also developing toolset based on experience.

Project: TBACS (NIST)

Starter: 1988: Kuhn's explorations (part of his job) of the potential role of formal methods in standards.

Booster: 1989: Identification of the smartcard application as a good one for actual use and agreement from NIST management and smartcard developers to perform the experiment.

Status: Formal methods were not continued directly but are playing a role in further encryption standards (transformed as state description and analysis).

Project: Tektronix

Starter:	1986: Research group became interested in formal methods, specifically Z.
Booster:	1988: Tek labs needed common architecture for oscilloscopes and research group needed applications to justify continuation during industry decline.
Status:	Z no longer used (in favor of object library) but still of interest; Tek waiting for another similar application.

Project: TCAS

Starter:	U.S. Federal Aviation Administration seeking better requirements for already deployed and troublesome system. Brought in Leveson as consultant, who was looking for formal methods demo project, as well as safety analysis subject.
Booster:	Adaptation of statecharts/predicate calculus into more readable notation. Accomplished by corps of graduate students and U.S. Federal Aviation Administration committee willing to accept new approach.
Status:	Collision Avoidance System Logic specification and pseudo-code undergoing (through end of 1992) Independent Verification and Validation. Safety analysis research continuing, in doctoral theses.

Project: Transputer (INMOS)

Starter:	Difficult problem (implementing floating point operations in software for T400) coupled with influence from Don Good lecture to Royal Society in 1985 on hardware verification.
Booster:	1986-87: Success with T414 floating point formalization showed way to verify hardware Floating Point Unit in T-800.
Status:	Now used as part of design verification process.

Project: Hewlett-Packard Medical Instruments

Starter:	1988: Applied Methods Group built up at Bristol, looking for applications. 1989-90: Several precursor projects with Medical Products Group in Oregon.
Booster:	1990: Waltham developer interested in trying formal methods and Bristol ready with process (Rigorous Software Engineering) and language (Hewlett-Packard Specification Language). Business concerns about U.S. Food and Drug Administration regulations are pending.
Status:	Failure of technology transfer: could not meet Waltham needs for faster time-to-market and improved quality.

8.1 Starter

The case studies suggest two prime reasons for deciding to use formal methods: technology explorations and recognition of difficulties. From the exploration perspective, key individuals in some of the organizations (e.g., Kuhn at NIST, Ladeau at Hewlett-Packard, Mills at IBM, Chapront for SACEM, May at INMOS) became interested in applying formal methods. The interests arose for a number of reasons, including pedagogical, demonstrating research ideas in a practical problem area, and transferring new ideas into their organizations. First encounters with formal methods came at several professional society events (e.g., with the presence of Tony Hoare), others through the literature, some from courses, and some as an outgrowth of larger research programs.

The second reason, recognition of difficulties, relates to concerns of assurance or productivity. The main cases supporting the latter view are DNGS, SACEM, and TCAS, cases that are safety-critical where regulatory bodies needed to be convinced of the safety of the products being developed. Comprehensibility was essential for assurance, with strong need for intellectual control. Commercial cases also manifested similar needs for intellectual control, but with objectives focused on long-term process improvement, (e.g., CICS restructuring, Cleanroom quality goals, Tektronix product framework, INMOS reduction of testing, HP general quality goals).

Projects such as LaCoS and MGS are a blend of the two objectives, with government support as the agents for creating the projects.

8.2 Booster

An analysis of the data does not lead to one set of common reasons that the use of formal methods was boosted. In the HP and TBACS cases, a product development was used as a vehicle for experimenting with formal methods. In both instances, the risk of using formal methods was extremely low in the development schedule, yet had potential benefits pedagogically and for product quality.

There were cases (e.g., INMOS, Tektronix) where difficult technical problems were recognized (e.g., understanding a scheduler, developing reusable platforms) and the mathematical analyses supported by the formal methods were viewed as a means of solving the problem.

In other instances, successful applications of formal methods (e.g., CICS, Cleanroom, SSADM) have led to the ongoing application of the concepts and some diffusion of the ideas to other parts of the organization.

Technology sometimes acted as a booster (e.g., TBACS, SACEM, INMOS, MGS) but other times was a retardant (e.g., HP). In several cases, technology was not a key factor (e.g., CICS, Cleanroom, and Praxis) and more technology might have accelerated or retarded the case progress.

Finally, one must not underestimate the power of contracts and government programs. Government contractual requirements (e.g., MGS) or government programs (e.g., ESPRIT) can play a role in diffusing the technology into commercial organizations.

8.3 Current State

In the majority of cases, the use of formal methods is continuing. However, one should note the couple of instances where it is not continuing. The Hewlett-Packard group at Waltham decided not to proceed with the use of formal methods as it did not appear to bring sufficient benefits; there was no improvement in quality and no reduction in development time. Hewlett-Packard has disbanded HP-SL and Applied Methods Groups. This suggests that the production divisions were not developing any convincing evidence to support continued work in the area. Reasons pertaining to notation and a different "mindset" were given for the lack of transition. Tektronix, however, is different. In that instance, there were clear benefits in applying Z that led to the desired software architecture. However, the Z work was transformed into object-oriented code and it is from that code that Tektronix is continuing their development. Clearly, in the engineer's and programmer's view, "code is the boss," not a mathematical specification.

The diversity of ways methods were used is striking. In some cases, notations were adopted "as is" and a process built around them, but in other cases the methods evolved significantly. Similarly, the organizations are evolving as their processes assimilate formal methods.

The regulatory cases are individually and collectively leading to better understanding of the role of formal methods in standards: what they might achieve and how they might be used. Thus, current efforts in standards are being influenced. Some companies (IBM Cleanroom, Praxis, and GEC Alsthom) are using their maturity in formal methods as evidence of "best practice" in building their products.

8.4 Timing

The data do not suggest any consistent perspective on timing issues. Some organizations (NIST, HP) were able to perform their experiments reasonably rapidly and draw conclusions. The regulatory cluster cases required substantial labor to complete their work. It appears that where changes or development of standards are required that there is substantial effort required in inserting the new technology.

As a gross generalization, we might characterize a long-term process for assimilating formal methods as going through three stages:

- Years 1, 2, 3: exploring different methods and applications
- Years 4, 5, 6: serious experimentation, including measurement, on commercial or regulatory products
- Years 7, 8, 9: development of appropriate tools and processes around successful methods (from phase 2) in the style of the organization and its business needs

Some organizations falter in the first phase through lack of resources or wrong choices of methods and applications. For those that made it through phase 1, we see some that are in phase 2 and some in phase 3. A phase 4 would probably include greater interaction of formal methods with other methods and greater promotion throughout other units of an organization.

9. ANALYSIS OF FORMAL METHODS R&D SUMMARY

In this section, we focus on topics of specific interest to the formal methods R&D community, with the purpose of identifying some general themes from the case-by-case formal methods R&D summary.

For this section, we will fold the cases from the exploratory cluster into the regulatory and commercial clusters. The exploratory cases differed more in their application and organizational context than in their use of formal methods. Consequently, we will view TBACS as part of the regulatory cluster; and HP and LaCoS as part of the commercial cluster.

9.1 Regulatory Cluster

Figure 6 presents the R&D summaries for the four regulatory cluster cases and TBACS[8].

Before beginning our discussion of the regulatory cluster, a few background comments are necessary to set the context for regulatory cases. In the regulatory arena, government agencies are involved in certification. Such agencies may chose to certify the product being developed, the process by which a product is developed, or the individuals involved in product development. In our case studies, MGS and SACEM were a mix of the first two approaches. TCAS and TBACS are product certification. DNGS is also product certification, since AECB was uncomfortable with the processes being used.

---

[8] In these figures, the "needs" are as described by the developers.

Project: DNGS (Ontario Hydro, AECL, AECB)

Formalisms: Software Cost Reduction style specification, program function tables describing functional behavior of code routines

Uses:

- Specification of "systems," not solely "software."
- Program function tables are a tabular representation of a Mills-style functional description of a routine.
- Specifications may be an object of review; part of the documentation.
- Proofs that a program function table description of a routine is consistent with the routine's specification.

Tools:

- Microsoft Excel was used to manipulate some of the tables.

Needs:

- Tools for automated deduction, developing functional descriptions of code routines, bookkeeping.

Fig. 6(a) — DNGS R&D Summary

Project: Multinet Gateway System (Ford Aerospace)

Formalisms: Gypsy and Trust Domains

Uses:

- Gypsy specification and programming features. Specifications are in terms of a first-order typed predicate calculus; imperative programming language.
- Trust domains for describing networks (links and nodes).
- System security properties described in terms of information flow.
- Proofs that security properties are met.

Tools:

- Gypsy Verification Environment (GVE)
- Extractor tool to find minimal information needed for a proof.

Needs:

- Improved automated deduction.
- Better capabilities, in the GVE, to handle industrial-scale projects.
- Soundness demonstration of GVE proof system.
- Improved specification language expressibility.
- Notation more acceptable to engineering staff.

Fig 6(b) — Multinet Gateway R&D Summary

Project: SACEM (GEC Alsthom, Matra Transport, Compagnie de Signaux et Entreprises Electriques)

Formalisms: After starting with Hoare assertions, the B Abstract Machine approach has been adopted.

Uses:

- Abstract Machines are specified for the various components of the system being specified.
- Concurrency and timing are dealt with separately.
- Refinement is defined and used.
- Newly developed tools are capable of generating code from a low-level specification.
- Verification is performed at the top-level and lower by proving invariants and refinement obligations.
- Validation activities include extensive simulation and operational scenario generation.

Tools

- Early on, a substitution processor was developed for Hoare assertions.
- The B tool is the base for both theorem proving and specification management.
- Based on SACEM experience, the FORSE toolset is being commercialized to support the methodology used.

Needs

- Better integration of formal specification and proof with other validation activities, at least conceptually.

Fig. 6(c) — SACEM R&D Summary

Project: TBACS (NIST)

Formalisms: State transition model with assertions expressing the security policy. Language used was Ina Jo.

Uses:

- C code was the base language for the simulator from which the state transition model was derived. It was matched with the security policy.
- The spec was used for informal communication with the smartcard designer.
- Errors were discovered and the experience gained was used to improve the manufactured version and the overall design.

Tools

- Selected functions were verified using the Formal Development Methodology theorem prover.
- External paper-and-pencil cross-reference were used.
- Scrolling was used to overcome interface problems.

Needs

- Better understanding of this reverse engineering process into a state transition specification is worthy of investigation.
- Improved interface for managing large expressions and long series of proof steps.

Fig. 6(d) — TBACS R & D Summary

Project: TCAS

Formalisms: A modification of statecharts, state machines

Uses:

- Handles concurrency as parallel state machines.
- Tabular notation (embodying Disjunctive Normal Form) for transition conditions.
- Specifications subject to review and Independent Verification and Validation.
- CAS Logic formalism being determined from pseudo-code and English.

Tools:

- LaTeX

Needs:

- Safety analysis tool.
- Automated deduction and model checking support.
- Well-formedness checker.
- Development and description of underlying mathematics.

Fig. 6(e) — TCAS R & D Summary

Note that certification is also a *process*. Currently, there is substantial interplay between researchers, developers, and certifiers. What is unclear at this point is whether formal methods will be used for product certification or as process evidence.

9.1.1 Methods

Frameworks. First-order logic was used on all the projects. As well, not surprisingly, state machine formalisms predominated. For example, in SACEM, the B Abstract Machine approach clearly incorporates the "state machine" perspective. Similarly, the Formal Development Methodology (TBACS) explicitly used state machine representations for specifications; the specification is defined in terms of state transitions and invariant properties. We felt that most individuals involved with these cases were comfortable with the "state machine" mindset. Code proofs generally followed the Hoare Logic approach or a closely related method (strongest post-conditions and equivalence with specifications). Abstraction and "blackbox" specifications were fundamental to the cases.

Note that, even though there are different methods being used (Software Cost Reduction specifications and strongest post conditions; Gypsy; Hoare Logic and B; Statecharts; and Hoare Logic for the programming languages associated with the Formal Development Methodology), there are strong similarities with the underlying mindsets for specification and proof. What appears to be happening here is that essentially the same ideas are being packaged in different ways.

Code proofs and refinement. A distinguishing characteristic of the regulatory cluster is the demonstration that code is in conformance with requirements, correspondence demonstrated by the presence of code proofs and/or refinement in three of the cases. Conformance is also manifested, in some instances, by substantial effort. A particular example is the DNGS case. Here, it was clear that the regulator (AECB) had substantial concerns about the software. As a consequence, a labor intensive activity (as there was no tool support) proceeded in which there was manual justification that the code (written in Pascal, FORTRAN, or assembler) was equivalent to the requirements. Three independent teams were involved: (i) the requirements team (who reverse-engineered the system requirements); (ii) the code team (who developed the proof function tables); and (iii) the proof team (who were responsible for the equivalence proofs). $4 million (Canadian) has been attributed to the verification effort and, at one point, approximately 30 individuals were involved in different aspects of the verification. One can note similar traits with SACEM, where, even prior to the use of B, they were performing code proofs using Hoare Logic. For SACEM, it appears that around 120,000 hours were used on Verification and Validation efforts (the 120,000 includes analyses in addition to formal methods). Yet, the amount of code for the DNGS and SACEM cases was quite small; though the specifications for embedded systems can be complex. For one of the DNGS shutdown systems (SDS1) there were about 2,500 LOC (FORTRAN and assembler); for SACEM, there were 9,000 LOC.

Note that these code efforts are in addition to the use of formal methods to remove or inhibit the addition of errors to requirements and specifications. In the regulatory cases, formal methods were not used solely to capture requirements; they were also used to assure that code conforms.

Communication. Effective communication of specifications and requirements to individuals not knowledgeable in formal methods was an important characteristic of the regulatory cluster. TCAS and MGS provide cautionary tales for those of us who are familiar with mathematical notation and, perhaps, no longer recognize the potential impediments of improperly chosen notations.

With TCAS, Leveson and her group chose a notation that was a simplified form of Statecharts[9], augmented with tabular representations of Disjunctive Normal Form predicates describing state transitions. The combination of graphical notation and "decision tables" that were close to the engineering notations used by the engineers on a TCAS committee provided an effective communication milieu. Earlier efforts that used common predicate calculus notations were impediments and were unsatisfactory. Similarly, with the MGS, there was difficulty using Gypsy to communicate with the developers; instead, a graphical notation was provided.

The importance of language design and the ease with which the underlying semantics can be described cannot be underestimated in developing a successful formal method. Developing computer-controlled systems are already complex activities; we do not need to augment the problem with needlessly complex languages.

9.1.2 Tools

Tool impoverishment. From the perspective of the early nineties, one must view the degree of tool use to be impoverished. Mostly, this is a result of the newness of the technologies being used and, perhaps, a lack of recognition of what tools are available. Figure 7 summarizes the formal methods tools used in the cases. Neither TCAS nor DNGS made any use of formal methods tools. We have already noted above the effect this had with DNGS. It is the view of the authors that much of the tool support needed for DNGS already existed; however, the developers did not have the time to investigate availability or to modify the tools that might have been used. With TCAS, the language being developed is still so new that it has not yet been an object of tool development. However, the one tool TCAS did use, LaTeX, was in support of their primary goals of readability and reviewability of the specification. Careful consideration was given to the presentation of the specification and for cross-referencing between related concepts.

SACEM used the B tool and MGS the Gypsy Verification Environment to handle proof and specification. With respect to the MGS, the use of the Gypsy Verification Environment was necessary because of the U.S. National Computer Security Center mandate to use an "endorsed tool" to achieve an A1 certification. From our current perspective, the proof checking capabilities of the GVE are weak

---

[9] A simplified subset was chosen since "broadcast communication" was not appropriate to their modelling approach and they did not require the additional modelling approaches provided by Statecharts. Consequently, the STATEMATE toolset was not usable.

Regulatory	
DNGS (SDS1)	None
MGS	Gypsy Verification Environment
SACEM	B
TCAS	None
Commercial	
SSADM	Prototype Z parser and typechecker
CICS	PS/2 based toolsuite w/ editor, type checker, semantic analyzer (Z)
COBOL/SF	None
Tektronix	fUZZ Z editor, typechecker and pretty printer
INMOS	Occam transformation system; in-house refinement checker
Exploratory	
LaCoS	RAISE toolset
TBACS	Formal Development Methodology
HP	HP-SL syntax checker

Fig. 7 — Formal methods tools used in cases

and are surpassed by a number of existing systems (primarily in North America). Furthermore, there were worries about the soundness of the prover, although the developers of the MGS have confidence in the GVE. (Similar comments may be directed at the Formal Development Methodology theorem prover.)

Soundness of automated deduction systems. If automated deduction systems are to be used in the development of critical systems, one must be able to demonstrate soundness. If the automated deduction system has complex heuristics and algorithms (especially if the algorithms are non-axiomatic, such as the use of Linear Programming techniques) then justification that valid proofs are being presented is necessary. One might take the view that one must have as much confidence in the toolset as in the application. One approach to demonstrating soundness is to separate the "proof discovery" process from the "proof certification" process. It is likely much easier to certify a "proof certification" tool than a full fledged automated deduction system.

9.1.3 Needs

Integration of validation techniques. We found from these cases that substantial work is required on how to best integrate formal methods with other assurance (validation) and software engineering activities. It appears that from the four cases in the Regulatory Cluster that the SACEM developers are on the forefront. They have developed a set of techniques that have not only been used on SACEM, but on two successor efforts (for the French railway, SNCF, and for a system in Calcutta, India).

Better automated deduction systems. The authors feel that the regulatory cluster of cases has demonstrated a greater need for automated deduction support, proof obligation generators and for refinement (from specification to code) than the other clusters.

Extension of formal methods capabilities. Formal techniques for handling concurrency, real-time, and asynchronous processes were suggested as areas of future research. Currently, other means (such as simulation) are being used. We are puzzled as to why those we interviewed were not particularly bothered by the absence of such formal techniques.

9.2 Commercial Cluster

Figure 8 presents the seven R&D summaries for the five commercial cluster cases. This is explained in the introduction to Section 9.

9.2.1 Methods

Z has been the primary framework. Our commercial cases predominately used Z, either to specify systems (CASE, CICS, and INMOS) or as a mathematical language to analyze and model various design choices (Tektronix). As with the regulatory cluster, the languages used are first-order and state-machine based. The language used with LaCoS, RAISE, is an exception in that it is a complex, wide-spectrum language that allows for different modes of use. A number of the cases did investigate refinement issues; however, as noted below, refinement has not been particularly important to industry.

We asked ourselves why Z currently appears to predominate industrial use of formal methods outside the security arena. In brief, the predominance appears to result from the choice of set theory; the development of an accessible (readable and writable) notation; reasonable support for state machine specifications; close interaction with industrial concerns; and a substantial pedagogical literature.

Limited application of code proofs and refinement. Unlike the regulatory cluster, there was not as much interest in refinement and code proofs. The principal exceptions to this observation were Cleanroom and INMOS. Proof, at least at the informal level, is fundamental to the Cleanroom methodology. INMOS used refinement (as supported by the Occam Transformation System) and proved correctness properties for finite state machines. A main reason for the reduced interest for refinement appears to be based on cost; i.e., value for money. The use of formal methods to formally describe requirements and specifications has been found to be beneficial with respect to improved quality and, in some instances, demonstrable reduction in costs.

Modelling. In the commercial cluster, formal methods have been used mainly for modelling and communication. The usual attributes of improved precision, conciseness, and mathematical modelling were important to all the cases. A modelling example is the oscilloscope work performed at Tektronix to develop a reusable software infrastructure. An example of the use of formal methods in communication is the use of Z in round-table discussions as clients and developers discuss the adequacy of specifications.

Processes. While process was important in all the cases, it was directly addressed only in the RAISE and HP cases. Interestingly, while HP-SL is not being adopted by the Waltham Medical Instruments Group, there is the intention to use the Bristol-developed Rigorous Engineering Methodology. The RAISE project has produced a Method Manual outlining their approach. Cleanroom has also emphasized process by separating the development and testing teams, and by the use of statistical testing to infer quality measurements.

9.2.2 Tools

Language checkers. The tool support used by the commercial cluster was generally limited to the use of language well-formedness checkers. Since four of the cases involved Z, the tools were generally Z editors, type checkers, pretty-printers, and cross-referencers. All the tools were prototypes.

Project: CASE (Praxis)

Formalisms: Z with English annotations

Uses:

- Z used for formal high-level specification of the infrastructure (management facilities) and some of the tools.

Tools:

- Prototype Z parser and typechecker from Forsite project (with added ASCII screen-based editor), plus troff.

Needs:

- For method, an ability to handle concurrency. For tools, a Z schema expander, enhanced editor, and browsing and x-ref facility.

Fig. 8(a) — SSADM R&D Summary

Project: CICS (IBM)

Formalisms: Z with English annotations

Uses:

- High-level specification of CICS "basic services" modules.

Tools:

- PS/2 based toolsuite consisting of editor, typechecker, semantic analyzer, LaTeX macros.

Needs:

- For method, an ability to handle timing. For tools, a Z schema expander, true semantic analyzer (based on a complete Z semantics), a pre-condition calculator, and configuration management facility.

Fig 8(b) — CICS R&D Summary

Project: Cleanroom (IBM)

Formalisms: Functions and box structures (state machines) plus whatever other formalisms, (e.g., grammars) are needed.

Uses:

- Functions are decomposed and box structures have state structure added (black → clear).
- Specifications are extensively reviewed for completeness to stabilize requirements before design proceeds.
- Usage profiles for testing add context to specification.
- Verification is a review process lead by "correctness questions."
- A reliability model is used in testing to predict and control quality objectives.
- A continual objective is simplification of designs, data structures, algorithms, etc.

Tools:

- Simple editors are used for text management.
- Proof checkers are less desirable because Cleanroom emphasizes the review process and its mental group correctness checking.
- The most often used tool is the wastebasket for overly complex designs.

Needs:

- Tools for extracting and tracking verification steps are under consideration.

Fig. 8(c) — Cleanroom R&D Summary

Project: LaCoS (Lloyd's Register; Matra Transport)

Formalisms: The RAISE Specification Language, a wide-spectrum specification language supporting different styles (applicative, data types, imperative) and refinement with proof obligations. A prescribed process is also developed in a Method Manual.

Uses:

- Both organizations interviewed were using RAISE Specification Language for modelization: the development of a model for exploring requirements, but not necessarily serving as the base for design.
- Concurrency specification is needed but not considered satisfactory in the RAISE Specification Language.
- Refinement down to code is considered quite important, but difficult.
- Justification (i.e., the discharging of proof obligations) is being explored using tools.

Tools:

- The RAISE toolset is based upon the Cornell Synthesizer Generator, both as editor and for processing language objects. The editor capability is not widely used, but the Cornell Synthesizer Generator controls the semantics of the RAISE Specification Language.
- Automated reasoning tools are just becoming available, specifically a Justification Editor to permit experimentation with justification proofs using different styles and underlying support tools, such as simplifiers.
- Configuration and object management are supported.

Needs:

- Better understanding of the complex RAISE Specification Language.
- Experience with automated reasoning tools.

Fig. 8(d) — HP R&D Summary

Project: Hewlett-Packard Medical Instruments

Formalisms: Rigorous Engineering Methodology and HP-SL

Uses:

- HP-SL is a VDM-like specification language.
- State machine and abstract data types.

Tools:

- HP-SL syntax checker.

Needs:

- HP-SL well-formedness checker.
- Notation which is more acceptable to engineering staff.

Fig. 8(e) — HP R&D Summary

Project: Tektronix

Formalisms: Z

Uses:

- Abstract specification of system operations and user interface in Z; Z specification translated into lower-level representation that is closer to implementation.

Tools:

- fUZZ editor/typechecker/pretty-printer.

Needs:

- Proof tool for refinement, precondition calculator, and schema expander.

Fig 8(f) — Tektronix R&D Summary

Project: Transputer (INMOS)

Formalisms: Z, Occam and HOL

Uses:

- High-level abstract specification of microprocessor components in Z and Occam; HOL used for proof checking of microcode-controlled hardware (still experimental).

Tools:

- Occam Transformation System; an in-house "refinement checker" that compares specification and implementation.

Needs:

- verified VHDL semantics and enhanced refinement checker to handle very large state spaces.

Fig. 8(g) — INMOS R&D Summary

No tools for Cleanroom. The two exceptions were Cleanroom and INMOS. Cleanroom accentuates the use of mental reasoning and has for the most part eschewed the use of tools. Fundamental to Cleanroom is group involvement and discussion of specifications. The group not a particular individual is responsible for delivered specifications and software. To a certain extent, the eschewing of tools echoes comments the authors have heard from researchers involved with the development of Z.

Transformations and correctness proofs for INMOS. In addition to using Z, INMOS used the Occam Transformation System and an in-house refinement checker. For example, with the Floating Point Unit, they used Z to specify the unit, manually derived an Occam "specification-level" description of the unit, and then used the transformation system to develop an "implementation-level" microcode description.

Unique in the commercial cluster, INMOS used an automatic "correctness-checking" tool for finite state machines on the T-9000 Virtual Channel Processor. It is of interest to note that this correctness-checking tool was limited to a decidable fragment of a theory of proof of correctness of state machines. By limiting themselves to a decidable fragment, it meant that T-9000 engineers could apply the tool automatically to their designs. It also had cost benefits in that some of the advantages of formal methods accrued, but that the engineers did not need to work through mathematical proofs. Such a tool becomes one of a repertoire that can be used by engineering staff.

Justification Editor. The LaCoS project is developing a "justification editor" for RAISE by which proofs about RAISE objects (e.g., code proofs) can be carried out. However, at the time of our survey, the editor was not in use.

9.2.3 Needs

Formal techniques for real-time. As with the regulatory cluster, there was interest in timing and concurrency. While INMOS, for example, made use of the Calculus of Communicating Systems and Communicating Sequential Processes to handle concurrency issues, simulators were still used to analyze real-time issues.

Better tools for Z. Perhaps because most of our cases had used Z, there were expressed interests in having various Z-related tools such as schema expansion. Other interests were for tools to navigate through specifications, such as browsing and cross-referencing tools.

Improved design methodologies. There was interest in developing design methodologies that encompassed formal methods technologies.

Notation and proof. Problems with notation arose from the HP case; there were difficulties finding people within HP who were willing to learn the notation and mindset. Improved automated deduction and additional experience with such tools were noted. There is also interest in gaining experience with the wide-spectrum RAISE language. Is such a large language a benefit or a detriment to using formal methods?

9.3 Overall Observations

In this final section of the R&D summary, we attempt to draw common themes from the 12 cases.

Need for Integrated Techniques. There is a clear need for improved integration of formal methods techniques with other software engineering practices. We have seen interest in such integration in at least two guises during our study: (i) inclusion of formal methods assurance techniques (such as Hoare Logic proofs) with other assurance techniques (e.g., hazard analysis, trajectory testing), and (ii) integration of formal methods with existing (or new) design methods (e.g., Jackson Structured Design). Successful integration is important to the long term success of formal methods. Industry will not abandon their current practices, but they are willing to augment and enhance their practices. So, the use of formal notations (like Z) to replace some informal notations such as English and pseudocode is an effective transition practice. But there are ancillary issues. We need to better understand how the assurance arising from formal methods analysis augments that from more traditional assurance techniques. Can we quantify such augmentation?

We found the INMOS and SACEM cases to be *important* examples of what can be achieved with integration (even if it is only in a nascent manner). The SACEM developers have put in place a set of techniques that they are now commercializing and applying to subsequent railway systems (such as a system in Calcutta and a country wide system for the French national rail system, SNCF); the SACEM developers have spent substantial effort in integrating various assurance practices. The INMOS case study is important for analytically demonstrating the power of mathematics. While in a number of applications of formal methods we were left with the opinion that informal notations are being replaced with formal notations (and there are clear resulting benefits), with the INMOS case study, our impression is that the power of mathematics has been applied in a nontrivial manner to design and analyze complex components (such as the Virtual Channel Processor). INMOS used formal methods in a strong engineering and opportunistic manner; when critical problems arose, they used the best techniques they could find to resolve the problems and, moreover, they refined some of the techniques so that they were cost effective in their domain of usage.

There are problems, however, with the large scale integration of techniques. For example, in critical applications, regulators need to understand what techniques are being used and what the limitations are.

Need for Cost effective tools. There have been formal methods tools in circulation since the seventies; yet few, if any, have found substantial application in industrial contexts. The limited dissemination of the tools is due to numerous facts including, but not limited to (i) the research and prototype nature of the tools, making them inappropriate for industrial usage, (ii) the steep learning curves associated with their use, and (iii) governmental limitations on usage. One cannot expect the average industrial firm to jump into an advanced formal methods tool based on algebraic data types, proof editors and mathematical logic. The firm will not have the staff educated for such a transition. In addition, we do not have in place the ability to estimate what the costs of application developments will be using these new techniques. Using existing practice, some measures for costing a project exist; such is not the case with formal methods where we know that the amount of effort spent at various stages of development is substantially different.

From the case studies, it is clear that industry wants rugged versions of specification language well-formedness checkers. In addition, these tools need to be integrated with other tools used in software development and should support configuration management, cross-referencing and browsing. Perhaps there is a role for Hypertext with some of these formal methods tools.

It is also clear, especially in the case of the regulatory cluster, that improved automated deduction support is essential. We are convinced that such support must have substantial automated support and there is a strong argument that checkers for decidable fragments of theories is one particularly fruitful area. Examples of such decidable fragments are the various model checking techniques (e.g., by Clarke's group at Carnegie-Mellon), finite state machine analysis (e.g., in the protocol area and as used by INMOS), and simplifiers that may be found in a number of automated deduction systems. INMOS made particularly effective use of a correctness tool for demonstrating equivalence between state machines, which did not require the engineers to have in-depth understanding of how to perform the logical reasoning. However, one must note that with such tools, and automated deduction tools in general, it is important to provide useful feedback when proofs fail. This is a sadly neglected area of automated deduction.

Currently, automated deduction tools do not provide the same amount of aid to software engineers as, perhaps, symbolic computations systems (e.g., REDUCE, MACSYMA, MATHEMATICA) provide scientists in their work, or finite state machine analysis tools aid protocol engineers.

Notational issues. One of the impediments to using formal methods with mathematical notations based on logic has been the difficulty in communicating with clients who are not comfortable with these notations. Various methods, such as animation, have been tried to overcome this hurdle. Perhaps we need to distinguish between the notations (and underlying ideas) that are used among formal methods experts and those that we use to communicate with individuals who are not expert in formal methods. This issue arose in a number of our case studies, with TCAS and MGS being prime indicators. In both cases, substantial effort was required to present ideas in a manner understandable to the *non cognoscenti*. Similar issues arose in the Tektronix and Hewlett-Packard cases. In Tektronix, the Z was replaced by object-oriented code. For the HP case, insurmountable problems arose because no one at the medical instruments group outside of the interested parties was willing or able to review the HP-SL specifications.

Other computing science trends. Of all our case studies, it seems that TCAS was the only one that made use of graphics. Yet technology advances in graphics, virtual reality, and multimedia are all progressing rapidly. How will formal methods evolve with these new ways of understanding and explaining our environment? Similar considerations arise with respect to CASE and object-oriented programming languages.

Difficulties of transition. No matter how positive we might want to be, we cannot say that the use of formal methods has taken industry by storm. There are a few organizations where the technology has been assimilated and is being applied fruitfully; but, for the most part, industry has been slow to assimilate formal methods. Why is this the case?

A number of potential reasons come to mind, and they are not unique to formal methods:

- a sophisticated technology is hard to transfer;
- new technologies need to be integrated with existing practice;
- new technologies have higher risk;
- there is a need for greater expertise; and
- some researchers are not concerned about transfer issues.

10. FINDINGS, OBSERVATIONS, AND CONCLUSIONS

10.1 Maturing of Formal Methods

Formal methods are maturing steadily in terms of

- the range of applications for exploratory, regulatory, and commercial use;
- the solutions to technical problems inhibiting larger scale use;
- the understanding of nontechnical barriers to wider spread use; and
- the standardization of concepts and notations.

Ten years ago, the popular impression was that proof of correctness of programs were generally applicable only to toy problems, but that some application domains (e.g., computer security) might warrant the investment in technology and know-how to scale up to real applications. In these case studies, we see two major changes over the decade:

1. The general concept of formal methods has enlarged from program verification (i.e., proof of correctness) to encompass formal specification, formal design, and other terms connoting the use of mathematical description and analysis in software and hardware development.

2. Organizations responsible for a wide range of applications have found it necessary to seek improved means of intellectual control over their complex system developments and have found ways that Formal Specification, Design, and Verification can be applied, along with other techniques, to meet those needs.

Our case studies cover the following kinds of applications: oscilloscopes (interfaces, operating systems, and data transformers), nuclear reactors (critical control component logic), trains (signalling systems, speed control, and stopping), aircraft (logic of collision avoidance), ships (engine monitoring), satellites (attitude control), smartcards (functions controlling network access), transaction processing, floating point arithmetic units, networks (security of access and data transfer), medical instruments (collection of monitoring data), programming environments (infrastructure and toolsets), and language processors (program restructuring).

The applications range from being quintessentially computer science (language processing, transaction processing, databases, arithmetic units) to complex interdisciplinary computer-controlled systems (reactors, aircraft, trains, and satellites). Some methods applied in many applications, although they were used in different ways.

The maturation of such a broad technology as formal methods faces both technical problems that could yield to more research and development and nontechnical difficulties that require organizational solutions. The technical problems addressed include

1. Expressing system functions — usually abstract state machines were used.

2. Following system architectures — language modularization techniques were adequate, although most could be improved.

3. Providing the most basic tool support — standard text processors were adapted, although only for stopgap measures.

4. Utilizing domain expertise — definitions, high level modelization, and hard intellectual work extracted the needed domain information.

Nontechnical barriers are now better understood and steps to overcoming them include:

1. Iterating with developers until notations become readable to nonformalists.
2. Identifying the necessary background mathematics, then developing and offering courses, emphasizing different aspects of the methods.
3. Starting to collect data and formulating models for calibrating processes using formal methods.
4. Starting formal methods applications without tools, through paper and pencil and minimal text processing.

Standardization steps include greater agreement on the common and most useful concepts:

1. Sets, sequences, mappings, graphs, grammars and other well-known mathematical structures suffice for describing many real system data structures.
2. State machines capture the basic behavior and structure of most systems, together with the above abstract data structures.
3. Multilevel specifications with interlevel mappings provide trails for many design processes.
4. Review is the primary assurance technique, with or without proving and testing.

It is interesting that the inherent mathematics is so basic, and could be understood by a talented high school student and certainly by undergraduates.

10.2 Scale of Application

Formal methods are being applied in system developments of significant scale and sometimes of extreme importance. The case descriptions in Volume 2 provide evidence for this claim, although the success of the projects relies on more than formal methods.

A strong lesson from these cases is the nature of scale. Lines Of Code (LOC), the most common measure, does not correlate well with scale in many of the applications seen. Not only is LOC an awkward measure (the question of formatting and comments, for example), but LOC does not assess several important aspects of systems:

- The power of abstraction that permits expression of a family of products in a single formal specification, apart from design and code, and, just as important, the communication that flows among stakeholders. (e.g., designers to managers, or among designers from different disciplines).
- The length and scope of a design and verification process — starting from safety studies, through many iterations of different design approaches, often across several organizational boundaries, culminating in what may be a few thousand lines of code deemed critical to system operation.
- The need for techniques to control (or at least reveal) the complexity of certain parts of systems.
- The possible loss of life or property for process control systems and the possible mayhem ensuing from unintended distribution of information across networks.
- The damage to reputation from products that are of poor quality or releases that are late due to too many system redesigns.

The case studies provide numerous instances of scale and manifestations of the above aspects, as seen in Volume 2.

10.3 Primary Uses of Formal Methods

The primary uses of formal methods in the cases studied are

Assurance: Providing high degree of confidence in systems with auditable information, and having targets of low or zero error rate. Cases: DNGS, MGS, SACEM, Cleanroom, INMOS, LaCoS, TBACS, and HP.

Analysis (of domain): Where formal methods are used to further understanding of the domain of interest (e.g., computer security as applied to communication networks). Cases: MGS, Cleanroom, Tektronix, INMOS, and HP.

Communication: Where communication between system stakeholders is the primary need. In this use of formal methods there is, perhaps, not as much mathematical analysis, but the replacement of informal notations with formal notations. Cases: TCAS, SSADM, CICS, Cleanroom, Tektronix, INMOS, and LaCoS.

Evidence of best practice: Where a competitive edge is acquired or where, as in "Assurance" above, formal methods are viewed as a necessary capability. Cases: SSADM, SACEM, and Cleanroom.

Re-engineering: Where a product is undergoing long-term upgrades and requiring recovery of structure or enhancement of function. Cases: TCAS, CICS, Tektronix, and TBACS.

Instances of these uses are:

- Applying formal specification at the internal interfaces of a large transaction processing system and using the specifications for documentation (CICS).
- Abstracting the commonality of a family of data monitoring systems to provide the platform for successor products (Tektronix).
- Reverse engineering a pseudocode system description into a more formally analyzable state and logic description (TCAS).
- Analyzing the security properties of a prototype system (TBACS).
- Using formal specifications to analyze the feasibility of a requirements statement and other applications through design (Praxis).
- Using extreme degrees of analysis to counterbalance lack of confidence in a development process and uncertainty about technology shifts from hardware to software (DNGS).
- Adopting formal specification and verification for added assurance in conjunction with other techniques (SACEM).
- Using specific tools and criteria to demonstrate process quality on a security-critical system (MGS).

- Using a comprehensive methodology to overcome technical challenges of a new product (Cleanroom).
- Evaluating an evolving methodology and toolset on a range of commercial and regulatory applications (LaCoS).

The role of communication among stakeholders is evident among designers, in the majority of cases, and between regulators and developers, notably TCAS and SACEM.

Note the diversity of ways that formal methods are manifested:

1. Specification only, to capture abstractions of product families or requirements models.
2. Specification with refinement through multilevel designs.
3. Proofs, sometimes of properties of specifications, of some refinement steps, or to a level close to code.
4. Proofs that use theorem provers and proofs performed mentally.
5. Reviews of specifications and design refinements, both the steps performed and other qualities (e.g., complexity and clarity) shown by the formal review process.

It has been observed that we did not include what might be the most obvious and simplest goal: cost reduction. Many uses in the long run do reduce cost, but this high level goal was not specifically stated in our interviews.

As would be expected, no single formal method (viewed as a notation and accompanying techniques) is sufficiently general to cover all of the significant characteristics of the applications addressed. There are many formalisms to choose from and currently more than one must be used to cover all characteristics. An important concern is that the use of several methods in concert is still not understood.

10.4 System Certification

Regulators are working with researchers and developers to develop practical and effective techniques for future system certification.

The four regulatory cases showed considerable interaction of different groups: certifiers, developers, and researchers. Recognizing the potential of formal methods to improve assurance within a regulatory process is only one step. Identification of the candidate methods seems to be quite opportunistic but the starting method then evolves to meet the needs of the various stakeholders. The long-term effect of these cases is seen in the standards that emerge from the experience with the process as well as the methods. While no standard has yet been sanctioned based on such cases, the de *facto* approaches may be determined by the experience and publicity of these cases.

10.5 Tool Support

Tool support, while necessary for the full industrialization process, has been found neither necessary nor sufficient for successful application of formal methods to date.

Flexibility, mixed modes of communication, and intellectual control have been the dominant drivers of those projects that reached successful conclusions using formal methods. However, several projects that have employed advanced tools (e.g., theorem provers) have not shown inordinate costs of use. The tools used in these cases are enumerated in Fig. 7 (Section 9.1.2).

The demands most strongly expressed were for simpler tools with "mass audience appeal." These include rudimentary editors, text processors, cross-referencers, and simple checkers. While many advanced tools (e.g., theorem provers) provide these capabilities, the cost of use is still very high. People have been conditioned by the quality, functionality, and documentation of $99 PC tools to expect comparable capabilities in all tools. More expensive and complex examples, such as CASE tools, also include many capabilities and make an effort to attract and satisfy users. Formal methods tools suffer by comparison with what is on the market for software development in general.

A limitation of the current formal methods support technology is the lack of common architecture that enhances scalable tool development. There are very large tools and very small ones, with not much in between. Few products enjoy any commercial support in terms of documentation, training, and evolution. Some internal company tools used in these cases may emerge in the marketplace within the next few years.

10.6 Technology Transfer

Several North American organizations and many more European ones have formal methods technology transfer efforts in progress. However, even those organizations with successful applications show only a small degree of penetration.

The following instances of defined technology transfer were seen:

- The LaCoS ESPRIT project is specifically designed to encourage technology transfer, centered around a specific tool and funded for several years.

- Cleanroom is supported by a Software Technology Center, chartered to perform training and demonstrations for other parts of IBM.

- HP sought a technology transfer route from its Applied Methods Group, but did not achieve critical mass and subsequent support within its developer shop.

- Many security applications have been successful but their results were, by virtue of their subject, not widely reported. Furthermore, many tools developed under the auspices of security organizations have been prohibited from transfer.

The following patterns were observed:

- Sometimes, greater success was enjoyed when formal methods were packaged as part of a larger methodology. In some cases, other aspects of the larger methodology were more successful than the formal methods. Formal methods often served as a "carrier" of modern and otherwise useful concepts.

- Successful cases usually had a "guru" or some form of sustained guidance.

- Management support at the right time was a key (e.g., at SACEM's critical review, within INMOS selection of design problems, in CICS's continuing relationship with Oxford, in Tektronix's commitment to pursue formal abstraction).

- Major technology change provided the impetus for adoption of formal methods in several cases (e.g., SACEM and DNGS faced change of technology from hardware to software; Tektronix required a common platform for its product family).

- There seem to be three general phases of maturity in organizations using formal methods: exploration, experimentation, and assimilation. Generalizing greatly, these appear to be three-year cycles, but there should be more analyses (including those of failed projects) to determine the progression of technology transfer.

10.7 Formal Methods Skills are Building

Skills are building slowly within organizations that are attempting experimental formal methods for use on industrial projects. The current educational base in the U.S. is weak in teaching formal methods in the context of software engineering.

The largest number of skilled persons using formal methods was within IBM's two projects: 40-50 at Hursley and about the same at Federal Systems Division. Most projects had fewer than 10 knowledgeable project members.

A theme discussed in Craigen and Summerskill [FM89], the lack of mathematical training in the general workforce, follows through in these cases. However, many engineering-oriented personnel were able to adapt to formal notations when presented to them in the right way. Given the elementary nature of the mathematics of most formal methods, the greater problem may be in the ability of formal methods users to properly model their systems and carry through designs.

The Carnegie-Mellon University's Software Engineering Institute began developing and disseminating a series of software engineering courses with a formal methods basis, but this effort seems to have tailed off in favor of the Software Engineering Institute process assessment emphasis. Courses are available at many major universities, but few are based on experience with cases such as these.

The European ESPRIT and U.K. Alvey projects have had an effect on the adoption of formal methods by supporting collaborative projects and featuring the industrialization of formal methods. Whether deemed successful or not, several projects showed direct or indirect effects of that funding.

10.8 Code Level Application of Formal Methods

Formal methods have been applied in a few instances at the code level of system development, but there are limits to applying formal methods at the code level.

Most programming languages lack adequate semantic bases to support the full application of formal methods (as so do many specification languages). Static analysis and compiler-type tools that have gained confidence may be used to complement design refinements carried close to a code level. However, many aspects of run-time environments and hardware (e.g., real-time and other performance aspects) remain to be treated adequately and must be viewed as holes in the general use of formal methods.

Refinement has not been seen as cost-effective for commercial applications although it may be more useful for regulatory cases.

10.9 Inadequate Cost Models

Metrics for cost and cost-benefit analysis are inadequate for capturing the differences the use of formal methods make in the development process.

Some organizations do have process descriptions and intuitive cost models in which they have sufficient confidence. It has been difficult to establish the value of formal methods in situations where already high quality can be otherwise achieved or where time-to-market dominates the product goals.

Some of the cases we studied attempted to collect performance data using accepted metrics (SSADM, CICS, SACEM, and HP). We found these to be inadequate in two respects. First, the metrics were those that are typically used in industry; these metrics rely heavily on comparisons with previous data collected using the same metrics, which had not been kept (the exception to this was CICS). Second, because formal methods, as practiced by most of the cases, require substantial change to the development process as compared to other, nonformal methods, the metrification methods did not seem calibrated to take these "process" changes into account. As a result of these inadequacies, we were unable to assess the data that was collected to reach meaningful conclusions. The most one can say is that metrics is an area for more research.

REFERENCES

[A792] T. Alspaugh, S.R. Faulk, K.H. Britton, R.A. Parker, D.L. Parnas, and J.E. Shore, "Software Requirements for the A-7E Aircraft," NRL/FR/5530-92-9194, Aug. 1992.

[Abr91a] J.-R. Abrial, "Introduction to Set Notations," manuscript draft of July 1, 1991.

[Abr91b] J.-R. Abrial, "Abstract Machines," manuscript draft of July 1, 1991.

[Abr91c] J.-R. Abrial, "A Refinement Case Study," manuscript draft of January 9, 1991.

[Arc90] G. Archinoff, R.J. Hohendorf, A. Wassyng, B. Quigley, and M.R. Borsch, "Verification of the Shutdown System Software at the Darlington Nuclear Generating Station," International Conference on Control and Instrumentation in Nuclear Installations, Glasgow, May 1990.

[Bea91] S.P. Bear, "An Overview of HP-SL," VDM 91 Formal Software Development Methods, Springer-Verlag, Berlin, 1991, pp. 571-587.

[Blo92] R. Bloomfield, personal communication, July 1992.

[BM91] T.B. Bollinger and C. McGowan, "A Critical Look at Software Capability Evaluations," *IEEE Software* 8(4), July 1991.

[BSI91] British Standards Institute, "Z Standard," Preliminary Draft, May 1991.

[CD*92] M. Carnot, C. DaSilva, B. Dehbonei, and F. Meija, "Error-free Software Development for Critical Systems using the B Methodology," in Proceedings of the Third International Symposium on Software Reliability Engineering, IEEE Computer Society Press, 1992.

[CGJ91] P. Clements, C. Gasarch, and R. Jeffords, "Evaluation Criteria for Real-Time Specification Languages," NRL Memorandum Report 6935, Feb. 1991.

[CZ90] S. Cardenas and M. Zelkowitz, "Evaluation Criteria for Functional Specifications," in Proceedings of the 12th International Conference on Software Engineering, Nice, France, 1990.

[deB80] J. de Bakker, *Mathematical Theory of Program Correctness* (Prentice-Hall, Englewood Cliffs, NJ, 1980).

[Dij76] E. Dijkstra, *A Discipline of Programming* (Prentice-Hall, New York, NY, 1976).

[Dil90] A. Diller, *Z: An Introduction to Formal Methods* (John Wiley & Sons, New York, NY, 1990).

[Dye92] M. Dyer, *The Cleanroom Approach to Quality Software Development* (John Wiley & Sons, New York, NY, 1992).

[EK85] S. Eckmann and R. Kemmerer, "INATEST: An Interactive System for Testing Formal Specifications," *ACM SIGSOFT Software Engineering Notes* **10**(4), 17-18, 1985.

[Flo68] R.W. Floyd, "Assigning Meanings to Programs," *Proc. American Mathematical Society Symp.* in *Applied Mathematics* **19**, 19-31, 1967.

[FM89] D. Craigen and K. Summerskill (eds.), "Formal Methods for Trustworthy Computer Systems (FM89)," in *Workshops in Computing* (Springer-Verlag, New York, NY, 1990).

[Gre91] K. Greene, M. Bouler, D. Jamsek, S. Gerhart, and T. Ralston, *Z: Notation & Use, Formal Methods Transition Study, MCC, STP-FT-3 (revised), Final Report*, August 31, 1991.

[Gri81] D. Gries, *A Science of Programming* (Springer-Verlag, New York, NY, 1981).

[GVE90] R. Akers, B. Hartman, S. Smith, M. Taylor, and W. Young, "Gypsy Verification Environment User's Manual," Technical Report 61, Computational Logic Inc., September 1990.

[Har87] D. Harel, "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming* **8**, 231-274, 1987.

[Hen78] K. Heninger, J. Kallendar, D. Parnas, and J. Shore, "Software Requirements for the A-7E Aircraft," NRL Memorandum Report 3876, November 1978.

[HP91] *Hewlett-Packard Journal*, December 1991, pp. 24-65, seven articles describing HP-SL and how it is used.

[Hoa69] C.A.R. Hoare, "An Axiomatic Basis for Computer Programming," *CACM* **12**(10), 1969.

[Hoa85] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall International, Series in Computer Science, (Prentice-Hall, New York, NY, 1985).

[HC91] W.S. Humphrey and W. Curtis, *Comments on "A Critical Look," IEEE Software* **8**(4), 1991.

[JLHM91] M.S. Jaffe, N.G. Leveson, P.E. Heimdahl, and B.E. Melhart, "Software Requirements Analysis for Real-Time Process-Control Systems," *IEEE Trans. Software Engin.* 17(3), 1991.

[Kem86] R.A. Kemmerer, D.R. Musser, D. Cooper, D.I. Good, K. Lovett, and D.H. Craigen, "Verification Assessment Study," Technical Report C3-CR01-86, National Computer Security Center, March 1986.

[Lee91] L. Lee, *The Days the Phones Stopped: How People Get Hurt When Computers Go Wrong* (Donald I. Fine Inc., New York, 1991).

[Lev91] N. Leveson, M. Heimdahl, H. Hildreth, J. Reese, and R. Ortega, "Experiences Using Statecharts for a System Requirements Specification," Proc. of the International Workshop on Software Specification and Design, Como, Italy, October 25-26, 1991, pp. 31-41.

[LS92] R. Linger and A. Spangler, "The IBM Cleanroom Software Engineering Technology Transfer Program," SEI Software Engineering Conference 1992, C. Sledge (editor), Lecture Notes in Computer Science 640, Springer-Verlag, Berlin, 1992.

[NCSC85] "Trusted Computer System Evaluation Criteria," DoD 5200.28.STD, U.S. Department of Defense, December 1985.

[NHWG89] M. Neilsen, K. Havelund, K. Wagner, and C. George, "The RAISE Language, Method, and Tools," *Formal Aspects of Computing* 1, 85-114, 1989.

[Par90] D. Parnas, G.J.K. Asmis, and J.D. Kendall, "Reviewable Development of Safety Critical Software," International Conference on Control and Instrumentation in Nuclear Installations, Glasgow, May 1990.

[Ros84] A.W. Roscoe, "A Denotational Semantics for Occam," in S.D. Brookes, A.W. Roscoe, and G. Winskel, "Seminar on Concurrency, Carnegie-Mellon University, Pittsburgh, PA, July 9-11, 1984," Lecture Notes in Computer Science 197, Springer-Verlag, Berlin, 1984, pp. 306-329.

[Ros86] W.L. Roscoe, "The Laws of Occam," Oxford University Programming Research Group, Computing Science Laboratory, Technical Report, 1986.

[Sch80] J. Scheid, "Ina Jo: SDC's Formal Development Methodology." *ACM SIGSOFT Software Engineering Notes* 5(3), 1980.

[Sho67] J.R. Shoenfield, *Mathematical Logic* (Addison-Wesley, Reading, MA, 1967).

[Sof91] IEEE Special Issues in Formal Methods, IEEE Software, *Software Engineering*, September 1991.

[Spi88] J. M. Spivey, *Understanding Z, A Specification Language and its Formal Semantics* (Cambridge University Press, New York, NY, 1988).

[Spi89] J.M. Spivey, *The Z Notation: A Reference Manual* (Prentice-Hall, New York, NY, 1989).

[US89] "Bugs in the Program," Staff Study by the Subcommittee on Investigations and Oversight, Committee on Science, Space, and Technology, U.S. House of Representatives, September 1989.

[VDM91] S. Prehn and W.J. Toetenel (eds.), "VDM'91: Formal Software Development Methods," 4th International Symposium of VDM Europe, Noordwijkerhout, The Netherlands, October 1991, Lecture Notes in Computer Science 552 (Springer-Verlag, New York, NY, 1991).

Appendix A

BIOGRAPHIES OF AUTHORS

A1. Dan Craigen

Dan Craigen is the Director of Ottawa Operations for ORA Canada. ORA Canada is a subsidiary of ORA Corporation (formerly Odyssey Research Associates) of Ithaca, New York.

Dan Craigen has been involved with formal methods since 1979 when he joined I.P. Sharp Associates. Since that time he has played an active technical role in the development of the m-EVES and EVES verification systems, has participated in various surveys of formal methods technology (including the U.S. sponsored "Verification Assessment Study") and has been involved in organizing a number of formal methods workshops (including FM89 and FM91). Mr. Craigen has also worked for Computational Logic Inc. in work leading to the development of a verifiable subset of Ada, called AVA. Mr. Craigen has a B.Sc (Hons) and M.Sc from Carleton University, Ottawa, Canada.

Since May 1989, when the EVES group moved from I.P. Sharp to ORA Canada, Mr. Craigen has been responsible for all contracts let to ORA Canada and has continued to pursue his research interests.

Mr. Craigen is a member of the ACM and an affiliate of the IEEE society.

A2. Susan L. Gerhart

Susan L. Gerhart is the Division Director, Computer and Computation Research, in the Computer and Information Science Directorate at the National Science Foundation.

From 1985 through 1991, she was associated with the Microelectronic and Computer Technology Corporation (MCC) Software Technology Program and its Leonardo project. Before that, she was on the software engineering faculty of the Wang Institute of Graduate Studies, a member of the research staff at USC Information Sciences Institute, and on the computer science faculty of Duke University. Dr. Gerhart's professional service includes the IEEE Software editorial board; advisory committees for Argonne National Laboratory, the National Science Foundation, Texas Advanced Research funding programs, and the National Research Council; and the organization of conferences in formal methods, software engineering, and programming languages. She also founded Applied Formal Methods, Inc., in Austin, Texas. Dr. Gerhart is a graduate of Ohio Wesleyan University and received her Ph.D. from Carnegie-Mellon University.

For the past four years, Dr. Gerhart has worked on the assessment of needs for the integration of formal (mathematically based) methods into software systems engineering and the improvement of current techniques and tools to meet those needs.

She led the Formal Methods Transition Study for 13 MCC member organizations to provide a baseline assessment of the state of formal methods internationally and then a strategy for transfer into

North American companies involved in critical applications such as telecommunications and transportation.

A3. Theodore J. Ralston

Mr. Ralston is currently President of Ralston Research Associates, a sole proprietorship consulting firm in software engineering and European technology analysis based in Tacoma, Washington. Prior to starting RRA in January 1991, Mr. Ralston was the European Technology Analyst for the Microelectronic and Computer Technology Corporation (MCC). While at MCC, Mr. Ralston helped launch a new research project in formal methods in MCC's Software Technology Program. This project, which finished in December 1991, was oriented toward transition to industry of formal methods, tools, and processes.

Prior to joining MCC in 1983, Mr. Ralston was on the research staff at Stanford University's Center for International Security and Arms Control where his responsibilities included teaching, research, and developing a computer database on international security and arms control. Prior to Stanford, Mr. Ralston served for seven years on the professional staff of the U.S. Senate Select Committee on Intelligence. His duties included analysis of strategic and tactical weapons intelligence, verification of arms control agreements, intelligence community budget oversight and authorization, foreign political risk assessment, and legislation and executive orders pertaining to the United States intelligence community.

Mr. Ralston received his undergraduate degree in Pre-Medicine from the University of Washington in 1971 (B.Sc in Zoology and B.A. in History) and his graduate degree in history and languages (Russian) from Oxford University in 1974. From 1964-1966 Mr. Ralston studied computer programming and systems analysis in an IBM-sponsored program. In 1988, Mr. Ralston was appointed to the National Academy of Science's Committee on International Developments in Computer Science and Technology. He is a coauthor of "The Verification Challenge: Problems and Promise of Strategic Nuclear Arms Control Verification," and two studies of export-import control policies on computer technology: "Global Trends in Computer Technology and Their Impact on Export Control," and "Finding Common Ground: U.S. Export Controls in a Changed Global Environment," both published by the NAS.

Appendix B

FORMAL METHODS TECHNIQUES

B1. SOFTWARE COST REDUCTION (SCR)

The description presented here is based on our understanding of the work performed at Ontario Hydro (on the Darlington shutdown system DNGS). This work includes evolutionary enhancements to [Hen78] by Parnas, his colleagues, and others. [A792] updates [Hen78].

B1.1 How the Method Works

Representations Used

Tabular representations of software requirements (described in a blackbox manner) and of program functions were used. In the latter case, the tables are called "program function tables" and, essentially, are "strongest post condition" descriptions of a procedure. The strongest post condition explicitly states how a variable is modified by a procedure. "Linkage tables" are also included.

The tabular approach to describing software requirements is derived from the work performed at NRL [Hen78] under the Software Cost Reduction (SCR) project. For this reason, we call the approach "SCR-style." The derivation of program function tables and the use of proofs (described below) were not part of the initial work on the SCR.

Steps Performed

The Darlington project was generally a reverse engineering exercise. The code already existed when they developed the specifications and other tables. Proofs were required to demonstrate that a routine specification was equivalent to the program function description. Proofs were performed by hand, though they could be mechanized.

Code linkage tables identify how the outputs of a program function table may be used as inputs to other program function tables. It is through code linkage tables that all the physical inputs and program variables that have an effect on a physical output are identified. Similar linkage tables exist for the specification. Hence, linkage tables help to modularize the descriptions of the specification and code.

Interaction between asynchronous processes was viewed as interprocess I/O occurring via shared data objects. Each process had its own specifications and program function tables. Processes were analyzed separately and the process interaction handled separately from program function table analysis.

Tools Applied

No specific tools exist to support the SCR-style of documentation. Tools are expected to be developed.

Roles Involved

Separate teams developed the specification tables, proof function tables, and the conformance proofs. Errors found during the proof process were reported back to both the specification and proof function teams.

Artifacts Produced

The software design specification document was produced, and documentation demonstrating that the code conforms to the specification (including program function tables, linkage tables, and verification summary tables) followed.

B1.2 Observations

What the Method Achieves

Conformance between code and specifications was demonstrated. Specifications are reviewable and conformance raises assurance.

Limitations

These include the absence of a toolset and the still evolving nature of the technology.

Other Techniques Supported and Required

Testing and hazard analysis were performed as part of the Darlington project.

User Community

To our knowledge, the collection of techniques described here has only been used by Ontario Hydro and AECL for the Darlington Shutdown Systems. Parnas and his colleagues continue to enhance the technology and are considering the development of tool support.

B1.3 References

See [A792], [Arc90], [Hen78], and [Par90].

B2. B

B2.1 How the Method Works

Representations Used

The B method uses the Abstract Machine Notation which supports the description of state (variables of the abstract machine), an invariant (constraints and relations among variables), and operations (interface with outside world).

Expressions may be written as guarded commands and the semantics are expressed in terms of substitutions.

Steps Performed

We can view the use of B as a five-step process:

1. Express specifications in Abstract Machine Notation and prove consistency of a machine, i.e., that operations preserve invariants.

2. Compose machines using (i) semi-hiding (no change of state variables), (ii) renaming for multiple use of same machine specification, (iii) parameterization, (iv) promotion of operations from used machines to interfaces of using machines, and (v) indications of parallelism.

   All compositions may have proofs of additional invariants constraining above compositions.

3. Refinement of a machine is either through data refinement (changing the variables so that they are closer to the target machine) or algorithmic refinement (so that the operations are closer to target control structures).

   Prove that refinement relation is transitive and monotonic.

4. Implementation (removal of unbounded choice), preconditioning, parallelism (possibly by automatic translations) with further proof obligations.

5. Importation (in an implementation of separately defined machines).

Design is defined as the process of identifying appropriate decompositions, refinements, and implementations.

Verification is defined as the discharge of all proof obligations (largely that invariants are maintained).

No prescriptions are given as to determining the top level specification, the number or size of refinements, or other decisions.

Tools Applied

B provides:

— an analyzer that generates proof obligations (effectively, a Verification Condition Generator),
— a type checker,
— an animator that symbolically executes abstract machines,
— status through the presentation of the development path (including discharge of proof obligations),
— checkpoint, reset, and remake (replays of analyses and proofs after machine modifications), and
— a prover that is applied to proof obligations.

Roles Involved

No guidance is supplied for organizing people in the development process.

Artifacts Produced

B provides abstract machine specifications and their proofs, refinements and their proofs, and compositions and their proofs.

B2.2 Observations

What the Method Achieves

A complete development may be performed and recorded. Changes may be accommodated using the replay tools.

Refinement, implementation, and composition steps have precise notions of correctness and mechanical generation of proof obligations.

Tests may be performed using the animator.

The final implementation step may be partially mechanized for common languages (e.g., C and Ada) and for some specification constructs.

Limitations

No guidance is provided regarding (i) design decisions or their recording (ii) testing or inspection methodology, or in the (iii) presentation of specifications.

The toolset is still evolving and, while commercially supported, it has not yet been evaluated by many outside organizations.

Other Techniques Supported and Required

None are identified.

User Community

SACEM used B for top-level validation and the successor Calcutta application used B as its design method, including generation of code.

A few academic groups have reported use (e.g., Oxford University), as have some U.K. government groups.

The B method was developed by J.-R. Abrial, a French mathematician and consultant. BP International developed versions of the B tool and B toolkit, which is sold commercially by Edinburgh Portable Compilers, Ltd. In conjuction with BP, Abrial is developing further documentation support. These can likely be obtained from Edinburgh Portable Compilers, Ltd.

B2.3 References

See [Abr91a], [Abr91b], [Abr91c], [CD* 92].

B3. CLEANROOM

B3.1 How the Method Works

The Cleanroom Methodology is presented here as

Cleanroom = Formal Method + other techniques.

While the "other techniques" are perhaps more significant than the formal method, we characterize just the formal methods referred to in the IBM Cleanroom.

Representations Used

These included

- Functions and behavior specs, i.e., assertions expressed as combinations of functions and logic in natural language or appropriate domain terminology;
- Box structures (not described in Dyer but used by Mills) -- state machines with varying levels of revealed internal state, i.e., black box, state machine, and clear box (internal structure revealed);
- Program Description Language using structured programming constructs.

Steps Performed

Stepwise refinement of functions to subfunctions using Program Design Language and data abstraction. Tabular formats for function specifications may be used.

Design verification — answering of correctness questions, e.g., for a loop "Does the loop function equal firstpart followed by secondpart?", "Is loop termination guaranteed for any argument?", "Does the specification equal identity when the loop condition is false?" Verification may be performed in written or mental/oral form, with additional inspection for other flaws, e.g. violations of standards, performance limitations, etc.

Tools Applied

A verification-based inspection syntax analyzer, which generates script for asking correctness questions, including flagging missing or redundant parts, plus summary of design language constructs, was used.

Roles Involved

Developers.

Artifacts Produced

These included specifications, refinements, and scripts of correctness questions and answers (i.e., verifications).

B3.2 Observations

What the Method Achieves

A complete development process is carried out with all correctness questions.

Since inspection is a team process, multiple points of view and checks are applied.

Design and verification are expected to bring the product sufficiently close to zero defect that statistical reliability testing is meaningful.

Limitations

No guidance is provided regarding design decisions or their recording. In addition, personnel must have acquired skills and discipline to adhere to the process standards.

Other Techniques Supported and Required

As described in [Dye92] (Figure 1.4, page 17):

- Function and Performance Requirements Specification with Usage and Build Statistics
- Incremental Software Development with Correctness Verification not Unit Test
- Independent Integration and Test of Released Increments with Representative Statistical Usage Samples
- Demonstrated Function and Performance for Acceptance with Certified Software MTTF

Note that Cleanroom uses formal methods in combination with all the above and that the specific function-theoretic approach described here could presumably be replaced by other approaches.

User Community

Cleanroom has been defined by IBM Federal Systems Division and exported to other parts of IBM through its Cleanroom Software Technology Center. Harlan Mills operates a company in Florida that provides consulting and training in Cleanroom. A recent National Science Foundation sponsored workshop at Rochester Institute of Technology and courses at the University of Tennessee introduce Cleanroom to academic audiences.

B3.3 References

See [Dye92] and [LS92].

B4. FORMAL DEVELOPMENT METHODOLOGY (FDM)

B4.1 How the Method Works

Representations Used

The Formal Development Methodology uses the Ina Jo language for expressing specifications and requirements, and Ina Mod for writing program assertions. The primary modelling approach is to use abstract machines. Specifications are expressed in terms of a state and operations that may modify the state. Operations are described as state transition relations. Correctness requirements are described by

using "criteria" and "constraints." A criterion is an invariant on states; regardless of which operation has been applied, the criterion will be true of any new state. A constraint relates two successive states and thereby excludes some state transitions. All operations must be consistent with the specified criteria and constraints.

Steps Performed

The FDM process is one of the successive refinements of abstract machines. At the top of the hierarchy of abstract machines is the Top Level Specification that describes the requirements, using criteria and constraints. Successive refinements elaborate the state and may add functionality, but must be shown to be consistent with the Top Level Specification. Once the lowest level of the hierarchy has been written, the developer must relate the entities of that final abstract machine with program fragments. FDM, through the Ina Jo processor (see below), will automatically produce entry and exit assertions that define procedural requirements.

Tools Applied

FDM provides

- the Ina Jo processor for checking static semantics and generating correctness assertions,
- verification condition generators for various programming languages,
- the Interactive Theorem Prover (ITP) for assisting in the proof of correctness assertions and verification conditions, and
- an ITP post-processor for generating transcript files of completed proofs.

Roles Involved

Domain experts for specifying requirements. There have been claims that less skilled individuals are needed to perform proofs.

Artifacts Produced

These include FDM descriptions (using Ina Jo and Ina Mod) of system requirements and specifications, and proofs that specifications (at all levels of the hierarchy) and code are consistent with requirements.

B4.2 Observations

What the Method Achieves

FDM can be used to express requirements and specifications using the abstract machine approach. Requirements and specifications may be recorded and are linked through the development chain. Though the authors are not aware of what explicit verification condition generators are available, in theory the methodology can link the requirements right through to code.

Limitations

The Interactive Theorem Prover was initially developed in the seventies and has not been substantially improved. The Interactive Theorem Prover provides only limited automated support.

Other Techniques Supported and Required

A means for symbolically executing specifications is supported [EK85].

User Community

FDM is one of two verification systems endorsed by the U.S. National Computer Security Center for use in the development of security-critical applications. FDM has been used on a number of security-related applications by U.S. government contractors. (An exploratory example, using FDM, is included in Volume 2: TBACS.)

FDM is supported by Unisys, Santa Monica, California.

B4.3 References

See [Sch80], [EK85], and [Kem86].

B5. GYPSY VERIFICATION ENVIRONMENT

B5.1 How the Method Works

Representations Used

The GVE uses the Gypsy specification and programming language. The specification language is strongly typed, first-order Predicate Calculus. The programming language is a Pascal derivative that includes a form of modularization (scopes), concurrency (processes communicate through buffers), exception handling, and abstract data types.

Steps Performed

Most applications have focused on using the specification component of Gypsy for modelling security properties. For example, various applications have described security in terms of noninterference: an output to a process cannot depend on a process that has a higher security classification.

More generally, the GVE supports the expression of application domain concepts, proofs of derived properties, the use of domain concepts to specify programs, and the proving of programs consistent with their specifications (using a variant of Hoare Logic). These steps are iterative as one's increased understanding of the application domain, as the development proceeds, will result in modifications and additions to the domain theory and code.

Tools Applied

The Gypsy Verification Environment provides (modified from [GVE90])

- an interface to external text editors for creating and revising Gypsy text,
- a parser for checking the static semantics of Gypsy text,
- a verification condition generator for generating formulae adequate to establish correspondence of code and specifications,
- a mechanical proof checker for assisting in the proof of verification conditions or in the construction of the proofs of mathematical conjectures,

- an information flow analyzer for establishing certain information security properties of special Gypsy specifications, and
- other utilities such as a pretty-printer and an algebraic simplifier.

Roles Involved

No guidance is supplied for organizing people in the development process.

Artifacts Produced

These include Gypsy descriptions of domain theories, specifications, and code of a software system. Proofs of domain theorems and verification conditions (the propositions produced by the verification condition generator) are also produced.

B5.2 Observations

What the Method Achieves

The GVE can be used in two (interrelated) ways: application domain theory development and exploration, and the production of software proven to be consistent with its formal specifications. Gypsy has, at times, been used as a design language.

Limitations

Toolset does not include a compiler for Gypsy. The proof checker is based on a system developed in the seventies and, except for a substantial rewrite, has seen only a limited increase in functionality. The proof checker provides little automated support. In addition, the GVE does not support fully the use of the Gypsy concurrency and abstract data type features.

The GVE and the documentation, as produced by Computational Logic Inc., does not include guidance for recording design decisions or for inspection. However, some users of the GVE (e.g., the Multinet Gateway System) have incorporated their work with the GVE in methodologies through which they have worked towards an A1 certification.

Other Techniques Supported and Required

None are identified.

User Community

The GVE is one of two verification systems endorsed by the U.S. National Computer Security Center for use in the development of security-critical applications. The GVE has been used on a number of security-related applications by U.S. government contractors.

The GVE is supported by Computational Logic Inc.

B5.3 References

See [GVE90] and [Kem86].

B6. HOARE LOGIC

B6.1 How the Method Works

There are numerous variants of Hoare Logic; we present one perspective here.

Representations Used

Hoare Logic may be viewed as an extension of First-order Predicate Calculus [Sho67] that includes inference rules for reasoning about programming language constructs. For this discussion, suppose that q1 and q2 are program fragments, that q1;q2 is their composition, that P, Q and R are predicates about the program state.

In [Hoa69], Hoare introduced the notation P{q1}Q to mean that if the predicate P is true of the program state prior to the execution of the program fragment q1, then the predicate Q will be true of the program state on the termination of the program fragment q1.

As a simple example of a Hoare-style inference rule (also called a proof rule), consider the rule of composition:

$$\frac{P\{q1\}Q,\ Q\{q2\}R}{P\ \{q1;\ q2\}R}.$$

This rule states that for the statement "P {q1; q2} R" to be true, we must prove two related propositions: that "P{q1}Q" and "Q{q2}R" are true.

In a similar manner, we can specify an inference rule for an assignment statement:

$$\frac{P => Q[e/x]}{P\ \{x := e\}\ Q} \quad \text{(with proviso).}$$

The "with proviso" caveat is meant to capture the idea that there are limits to the application of this rule. In particular, the programming language expression "e" cannot have a side-effect. In addition, notation "Q[e/x]" means that all free occurrences [Sho67] of "x" are to be replaced by "e" (with a technical proviso that some renaming of variables may be necessary). This rule states that for "P {x := e} Q" to be true, the implication "P => Q[e/x]" must be true. Note that we have eliminated all references to program text. The basic idea of the rule is that if Q is to be true after the assignment statement and the only modification to the state has occurred by modifying the variable 'x,' then Q must be true with 'x' replaced by the expression 'e' (under the hypothesis that P is true).

In essence, we are saying that the following must be true:

$$P => (Q[e/x]\{x := e\}Q).$$

Steps Performed

We will demonstrate the use of Hoare Logic on a simple swapping program.

Suppose we are interested in implementing a program that swaps two integer values. Then a specification might be:

```
procedure swap (var x, y : integer)
      pre true
      post x = 'y and y = 'x
```

where x and y are defined to be integer variables and 'x and 'y refer to the values of the variables on entry to the procedure. This specification states that if the procedure swap is called from a state that satisfies the precondition (in this case trivially true) then the program will terminate in a state where the values of x and y have been swapped.

To implement this program, we use a rather nonstandard technique to exhibit how to calculate the proof obligation (also known as a verification condition) for the implementation.

Ignoring issues of finite representation of integers, we implement swap as follows:

```
procedure swap (var x, y : integer)
      pre true
      post x = 'y and y = 'x
begin
      x := x + y;
      y := x - y;
      x := x - y
end swap
```

We want to show that this program meets its specification.

Using the notation Pre('x,'y) for the swap precondition and Post(x,y,'x,'y) for the swap postcondition, we want to prove that

```
Pre('x, 'y) {x := x + y;
             y := x - y;
             x := x - y} Post(x,y,'x,'y)
```

is true. Using the aforementioned rules, this verification condition reduces to

Pre('x,'y) implies Post((x+y)-((x+y)-y),(x+y)-y,'x,'y).

Simplifying the arithmetic calculations, this becomes

Pre('x,'y) implies Post(y,x,'x,'y).

Expanding the definitions of Pre and Post gives us

true implies y = 'y and x = 'x.

But in the initial state 'y and 'x are the values associated with y and x, hence, the proof obligation reduces to true.

Tools Applied

Since the early seventies it has been known how to eliminate code fragments automatically and produce a set of predicate calculus propositions (the verification conditions). Such a tool is usually known as a verification condition generator.

Roles Involved

As is described in [Dij76] and [Gri81], which describe the use of weakest preconditions, specifications and loop invariants are the responsibility of the developers; verification condition generation may be mechanized. Proof support tools exist, but will always require human intervention.

Artifacts Produced

Hoare Logic sets the foundation for logical proof that a program is consistent with its specifications and, consequently, may be used in documentation to provide assurance of the correct workings of programs.

B6.2 Observations

What the Method Achieves

Hoare Logic provides a means of demonstrating that a program is consistent with its specifications.

Limitations

Complex language features result in complex proof rules; it took numerous iterations to develop the first valid proof rule for procedure invocation. In addition, experience has shown that proof rules are exceedingly difficult to get right; model-based proofs to demonstrate that proof rules are valid are a necessity. ([deB80] describes an approach to such a demonstration.)

Other Techniques Supported and Required

Hoare Logic is one of the mathematical pillars for program verification and formal methods. Other analytic techniques must be provided as part of a comprehensive methodology.

User Community

Hoare Logic, and its variants, are used in numerous formal methods tools (starting from the early seventies).

B6.3 References

See [Hoa69] and [deB80]. Related references are [Dij76] and [Gri81].

B7. HEWLETT-PACKARD SPECIFICATION LANGUAGE (HP-SL)

B7.1 How the Method Works

Representations Used

HP-SL is a specification language that uses data types, functions, and logic to describe properties of software systems. Types in HP-SL are a set of similarly structured values that permit the same set of operations (Fig. B1 shows the predefined types in HP-SL).

Name	Description	Values
Bool	Boolean, truth values	True, False
Real	Real numbers	0,3,142, 0.0023
Int	integers	5, 0, 99
Nat0	natural numbers from 0	0, 9
Nat1	naturals from 1	9, 1
Char	characters	"a," "\[newline]"
String	sequences of characters	"abc"

Fig. B1 — HP-SL predefined types

Thus, numbers and Booleans are distinct types, since arithmetic operators work on numbers but not on truth values, and logical operators (e.g., ∧, ∨, → logical implication, ↔ logical equivalence, etc.) work on truth values but not numbers. All the expected arithmetic and logical operators are present in HP-SL.

Types may be given additional names in HP-SL, which does not introduce a new type but simply gives another name to an existing type. More complicated types may be constructed from other types by using type constructors which are predefined in HP-SL. These are sets, sequences, and maps.

Sets are types that are used to model collections of data for which order and repetition are unimportant. Sequences are types used to model collections of data for which order and repetition ARE important. Maps are types used to associate values among types. In programming languages, maps are constructed using hash tables, association lists, and trees, whereas in HP-SL they are constructed using a special type constructor.

HP-SL allows creation of new types, using the "type" keyword. The most common use of "type" is to construct RECORD or UNION types. A RECORD type contains a value(s) of several other types, and a UNION type permits choices between alternatives.

Functions are present in HP-SL and are used to model computations and calculations. They can be defined in one of two ways: explicitly (gives a formula to calculate a result) or implicitly (gives a test or definition for a correct result).

HP-SL has special functions called "Relations" which are used for modelling operations on the system state and to identify functions that return a Boolean.

HP-SL is based on logic (both propositional and predicate). Logic is used to constrain values, types, and functions to specify properties without use of algorithms.

Tools Applied

There is an in-house toolset (called HP-ST) available from HP's Bristol Laboratory that provides support for type and syntax checking, as well as editors (an EMACS version and an older version with a syntax-directed editor). HP-ST can be interfaced to the HP Softbench environment which provides some configuration management. There are no semantic analyzers or refinement tools.

Roles Involved

A few specification writers and several reviewers with the ability to read HP-SL specifications were required.

Artifacts Produced

The primary artifact of HP-SL is the single specification document, which consists of a mixture of formal HP-SL and natural language.

The main representations are textual.

B7.2 Observations

What the Method Achieves

HP-SL is good for identifying errors that result from misconceptions in the model of a system. It is also good at determining and specifying relationships between levels of specification and design, and as noted in the case write-up, HP-SL specifications can be used for generating test cases.

Limitations

The lack of the requisite mathematical skills in some users of HP-SL, which limits their understanding of the underlying theory is a bottleneck. Also, as noted by some of the users, pragmatic features of the notation can be difficult to grasp (this is no doubt related in part to the first limitation, and it should be noted, is no different than for many of the formal methods discussed in this study).

Other Techniques Supported and Required

HP-SL can be interfaced to the HP Softbench programming support environment, which gives HP-SL users access to some of the common services for project management available in Softbench, such as configuration management and version control. In addition, linkage between Softbench and the Portable Common Tool Environment (a European standard programming support environment) is also possible.

User Community

There is no widespread industrial user community outside of HP, however, HP-SL has been made available to academic and research communities.

B7.3 References

See [Bea91] and [HP91].

B8. OCCAM/COMMUNICATING SEQUENTIAL PROCESSES (CSP)

Occam is a high-level language developed for concurrent programming on the Transputer by INMOS and Oxford University. It can also be used as a specification language. It follows closely the principles of Hoare's language for communicating sequential process, or CSP, hence this section will describe the basics of both CSP and Occam. Both Occam and CSP rely on program transformation, which means replacing (some part of) a program by another (in the same language) with the same 'meaning' (i.e., equivalence).

B8.1 How the Method Works

Representations Used

The basic concept in CSP considers a process as a mathematical abstraction of interactions between the system and its environment. Recursion is used to describe long-lasting processes. A second feature is the use of traces to record the sequence of actions a process has carried out. The abstract description is then given a more concrete explanation using algebraic laws, and the last step is the implementation.

The notation for CSP uses first-order logic symbols plus some additional symbols for traces, functions, etc.

Occam programming principles closely follow those in CSP, although the notation looks quite different. For example, Occam uses prefix instead of infix operators, and indention instead of parentheses.

```
Occam   CSP

SEQ     for (P; Q; R)
  P
  Q
  R
```

The published denotational semantics [Ros84] of a large subset of Occam owes much to the failures-divergence models of CSP. Two processes are identical if they are able to engage in the same sequences of communications with their environment, and if at each stage they are able to (nondeterministically) refuse the same sets of communications; and if, moreover, whenever they terminate successfully, they have the same possible combinations of values stored in their free variables. This abstracts away such notions as efficiency, timing, and amount of true concurrency.

The approach of operational semantics is to consider the space of possible states of an ideal machine implementing the language, and the transitions between them corresponding to atomic actions. The modern style is to present a natural deduction logic which again allows the behavior of compound terms to be deduced from the behavior of their components. Such a semantics is well established by R. Milner.

Occam does not have any distinct notations for pipes, subordinate processes, or shared processes. All required communication patterns must be achieved by explicitly naming channels. However, procedures may be declared with channels as parameters. For example, a copying process may be declared

```
PROC copy (CHAN left, right) =
      WHILE TRUE
      VAR x:
      SEQ
         left?x
         right!x:
```

Occam is intended to be implemented with static storage allocation on a fixed number of processors. Consequently, if a chain of n buffers needs to be constructed using an array of n channels and iterative form of the parallel construct, which constructs n-2 processes (one of each value of i between 0 and n-3 inclusive), the value of n must be a constant and recursive procedures are not allowed (because of the static storage allocation).

Tools Applied

There are no tools for CSP, in the strict sense. However, the Occam Transformation System developed by Oxford University's Programming Research Group is an automated tool to assist in carrying out algebraic transformations. As the programs to transform grow larger, however, and the transformations to perform become more complex, this process necessarily grows more time consuming and prone to error. This is an area where mechanization as a computer program can help.

The Occam Transformation System consists of a suite of routines in the functional language Standard ML, which implements an abstract syntax for Occam, together with a number of operations on it.

The most important operations are implementations of algebraic laws interpreted as rewrite rules and implemented as functions mapping processes to processes. The laws express such facts as the commutativity of the parallel operator, or elaborating the behavior of the parallel composition of two arbitrations. In addition to these laws, certain others have also been coded explicitly, either to shortcut long derivations for the sake of efficiency, or to avoid the use of induction which would be required in their proof from the basic axioms. They are nearly all at the same low level of complexity.

To operate on real programs, routines are required to parse and display textual representations of the processes, which are provided. In addition, on top of the simple process type, an abstract datatype maintains a stack of contexts, allowing the user to browse through a process, to concentrate attention on one part, and to apply laws there rather than to the whole program. One of the features of the browser is a facility to 'fold up' extraneous detail, so as to be able to see the general structure of a large program all at once.

All these features are available through a window-based interface for SUN workstations, with syntax-directed editor and user-definable keys. On other machines, the interface is via the standard 'read-eval-print' loop of the ML interpreter.

Although the basic laws have a minimalist quality, and a large number of applications in sequence are required to achieve any substantial transformation, the underlying functional programming system allows these to be composed by higher-order functionals. Recursive strategies may thus be encoded, which apply a transformation throughout a process, or otherwise work at a higher level.

A number of editing and pretty-printing tools for Occam have been developed by users in Europe.

Roles Involved

The primary roles are as a specification writer, since Occam can be used to write high-level abstract specifications, and as a programmer, since Occam is also the programming language for the Transputer.

Artifacts Produced

The main artifacts, consistent with the two primary uses of Occam indicated above, are a specification document consisting of Occam statements of transformations, and an Occam program implementing the transformations.

There are no metrics other than the number of transformations. The main representation is textual.

B8.2 Observations

What the Method Achieves

The main contribution of CSP/Occam is as a programming language for parallel processing, principally in the area of synchronizing communications.

Any memory location shared between two parallel processes is read-only to both of them, and the synchronizing events are provided by point-to-point communications.

Programs are built up from assignments, input ('\bw?') and output ('\bw!') processes, by combining them in sequence or in parallel, and by choosing between them by conditional expressions or arbitration between alternative communications. Rather than providing block structured syntax (as in, for example, Pascal and Ada), Occam structuring is expressed through textual layout; grouped processes have the same fixed indentation relative to the surrounding code.

A more recent variant of the language, Occam2, adds simple data-typing and communication channel classification. Both were conceived primarily as a very-high-level assembler for the transputer family of microprocessors, which make it easy both to combine many processors to cooperate on a single problem, and equally to support efficiently many processes time-sliced on an individual processor.

Limitations

Like many of the other methods/languages, timing constraints associated with real-time operations cannot be handled very efficiently (this is not to say they cannot be handled, but the main technique is to abstract them away).

Other Techniques Supported and Required

Occam as a specification language requires a mental adjustment to a mode of thinking and analysis using algebraic transformations.

Occam as an implementation language can be used with other formal methods, such as Z, Communicating Sequential Processes or Calculus of Communicating Systems. We are not aware of any reason why it cannot be used with object-oriented styles, or with such structured methods like Jackson or Yourdon, although we do not know of any examples of this.

User Community

There is a growing set of Occam users in Europe, mostly in the scientific computing community, which uses Transputer-based mini-supercomputers produced by several companies. In North America, the only users we are aware of (outside of academic research interests) are users of the Transputer-based Floating Point Systems machines.

B8.3 References

See [Hoa85], [Ros84], and [Ros86].

B9. RAISE

RAISE stands for Rigorous Approach to Industrial Software Engineering.

B9.1 How the Method Works

Representations Used

These include the following: wide-spectrum language with types, values, variables, operations, and processes; and development", which are derivations from high level specification to code with intermediate proof obligations.

Steps Performed

Development steps take one specification into another specification (or into the programming language of choice).

Development process consists of

- capture — planning the design
- formulation — doing development
    - removing under-specification
    - changing between styles (applicative/imperative, sequential/concurrent)
    - changing types
    - adding new definitions or axioms
    - extending generality
    - substitution
    - transformation
- verification
- validation — demonstrating properties

Guidance is provided for each of the above in the form of standard decisions and their realization in language steps.

Quality assurance in the form of checks is also identified.

Recording and monitoring are discussed.

The CORE requirements method is presented as a requirements approach leading to an initial specification.

Tools Applied

A database underlies the RAISE toolset to record the RAISE Specification Language structures, semantic relations among them, and development steps.

A syntax-directed editor with multiple windows and buffers, text output to LATEX.

Proof tools follow an editor model, with well-formedness checks, transformations, generation of proof obligations, and invocation of simplifiers.

Translation tools translate the RAISE Specification Language into C, Ada, or other implementation languages.

Roles Involved

No specific guidance is supplied as to organization of persons involved in the development process.

Artifacts Produced

These include specifications (via the editors), proof obligations and proofs (via the proof tools), developments (via database records), and configurations (derived from the database).

B9.2 Observations

What the Method Achieves

A complete development may be performed and recorded.

Development steps have precise notions of correctness and mechanical generation of proof obligations. Validation is defined as establishment of properties.

The final implementation step may be partially mechanized for common languages, e.g., C and Ada, and some specification constructs.

Detailed descriptions of development steps and a general overall process are provided (and evolving under experimentation).

Limitations

The toolset is still evolving. While commercially supported, it has not been evaluated by many outside organizations. The RAISE Specification Language is complicated with over 500 proof rules, which must be semantically well-defined, incorporated into a justification editor, and dealt with in proof obligations. Managing this level of semantic complexity challenges both language developers and users.

Other Techniques Supported and Required

The CORE requirements approach has been described for front-end analysis.

User Community

RAISE evolved from the VDM model-oriented formal approach. It has been supported by ESPRIT projects and the VDM Europe support organization. Primary distributor is Computer Resources International, formerly Dansk Datamation Center, in Denmark. The LaCoS project is the primary user community, under support from ESPRIT. A North American distributorship for RAISE has just opened.

B9.3 References

The standard description is [NHWG89]. Product documentation includes overviews, method manual, tool manuals, and RSL manuals.

B10. TCAS METHODOLOGY

At the time of writing, the University of California at Irvine safety research group had not finalized a name for their language and methodology. For the purposes of identification, we will use the term "TCAS Methodology."

B10.1 How the Method Works

Representations Used

The language used with the TCAS Methodology is built on a subset of the Statecharts [Har87] language. The primary reason for basing their work on Statecharts was to take advantage of abstraction and graphical facilities, and to use state-machine representations.

A principal concern in developing the TCAS Methodology has been the development of readable and, hence, reviewable notation. In particular, a tabular representation of Disjunctive Normal Form[10] formula is used to describe state transition predicates.

A further modification to Statecharts is to view communication between system components as occurring over directed communication links. A TCAS Methodology description may be viewed as a series of parallel communicating state machines. In TCAS, some system components are "own aircraft," "other aircraft," and "ground-based sensors."

Steps Performed

In general, the TCAS Methodology can be used to model the requirements for process control. As described in [Lev91], the goal of most control systems is to maintain a relationship between the inputs and outputs of a system, in light of disturbances to a process. A controller uses a model of the process it is controlling to determine what corrective actions are required. This model is called the "internal model" [Lev91].

---

[10] A formula is Disjunctive Normal Form if it is of the form

$$Q_1 \ldots Q_k \, [P_{1,1} \text{ and } \ldots \text{ and } P_{1,n_1}) \text{ or } \ldots (P_{m,1} \text{ and } \ldots \text{ and } P_{m,n_m})]$$

where $Q_1, \ldots Q_k$ denote either universal or existential quantifiers.

The internal model is iteratively improved during the development of the requirements as the understanding of the process improves. The relation between the real-world process state and the internal model of the process state is to be explicit in the requirements specification and must be reviewable.

The TCAS methodology uses blackbox behavioral specifications, "information exposure" (providing as much information as possible so that the context for state transitions is clear), and hierarchy.

Tools Applied

Except for using LaTeX to aid in the production of readable requirements documentation, no tools yet exist to handle the TCAS Methodology. Tools are expected to be developed.

Roles Involved

No explicit identification of roles. During the early TCAS specification and (in parallel) specification language development, Leveson and her students wrote most of the specification while the domain experts were mostly reviewers of the evolving specification. As the process continued, however, responsibility for the evolution and correction of the specification has shifted to the application experts.

Artifacts Produced

This includes requirements documentation in a (potentially) formalizable requirements language. Explicit effort has been directed at achieving readability of the notation and for cross-referencing between parts of requirements.

B10.2 Observations

What the Method Achieves

The aims of the TCAS methodology are to develop readable state-based requirements specification documents and that the requirements are formally analyzable for correctness and safety properties.

Limitations

There is no tool support and a formal underpinning for the language has not yet been developed.

Other Techniques Supported and Required

Current research is underway for developing test cases from TCAS Methodology specifications. The TCAS Methodology is to form the foundation for future work in analyzing correctness and safety properties.

User Community

The TCAS Methodology is new and is being developed by Nancy Leveson and her research group at UC Irvine. It has been used to capture TCAS requirements, but has not been used elsewhere.

B10.3 References

See [Lev91] and [Har87].

B11. Z

B11.1 How the Method Works

Representations Used

The conceptual basis of Z is typed set theory, and the method is oriented to constructing models. Text and graphical representations are used.

The basic elements of Z are types, sets, tuples, and bindings. There is no universal set to which all elements belong, but a universe of disjoint sets called types (which contain basic types and composite types). A set in Z is an unordered collection of different elements of the same type, and Z supports the concept of infinite sets. A tuple is an ordered collection of elements that are not necessarily of the same type. A binding is a finite mapping from names to elements, not necessarily of the same type.

The main representational form is the "schema" which is a set of bindings depicted in a special "axiomatic box" syntactical form including a signature (or Schema Name) and a property (made up of two parts - the declaration and axiomatic constraints).

```
_Signature
| Declaration
|________
|
| Constraints
|
|__________________
```

Steps Performed

Operations can be specified in several ways in Z. One way is through the use of "axiomatic descriptions," which are unnamed schemas that introduce one or more global variables, and constraints on those variables. These specifications are called "loose specifications" by Z practitioners. A specifier uses these to indicate a function or constant has certain properties without giving it a value.

As noted above, a schema is a form that is named and has two parts: a variable declaration and a constraint part. In this approach, a specifier first specifies the state space in one schema (by mentioning the state space schema's name) which describes an operation in this state space.

There is no notion of control in Z, just as there is no notion of control in first-order predicate calculus. This means the specification gives no information about the order in which the various operations are performed. A special purpose operator ";" is used in the schema calculus to join two schemas sequentially in order to represent state changes, but there are no other control constructs (such as "if...then" or "while...do") in the schema calculus that might be expected for specifying sequential systems.

The semantics of Z are based on a version of Zermelo-Fraenkel set theory that does not include the replacement and choice axioms. A fuller description of the Z semantics is currently under development by the ZIP project, which is a U.K. project to develop a Z standard.

Constructing models in Z does not require that the specifier adopt a point of view. While it is possible to write Z specifications that look like executable functional programs, finite state machines, or algebraic theories, most published Z specifications (carried out by Z proponents) are written from the "state-transition" point of view. These specifications typically begin with a statement of the base types of the system, followed by a state space description (using a schema), then an initial state description (another schema), followed by a series of operation descriptions interspersed with invariants, constraints, and exception handlers, all represented by schemas. In addition (and this is noteworthy given the experience reported in the Z cases in this study), natural language descriptions can be used to annotate Z schemas and often are an important part of a Z specification. For example, there is no formal notation for stating that a given schema is a state space description, an invariant, or an operation — this information is left to the natural language annotation.

Z specifications are usually refined as follows. If A1,A2,...,Ai are a collection of schemas defining an abstract system, a refinement might be another collection of schemas B1,B2,...,Bj that describes the system at a lower level, plus another collection of schemas C1,C2,...,Ck that describes the relationship between the two levels.

Given the "state-transition" point of view used by Z proponents, events, objects, and properties in most Z specifications are expressed in this context by using

- basic types declaration,
- a state space description,
- an initial condition,
- operations,
- invariants, and
- constraints.

With respect to constructing Z specifications, a Z specification is basically composed as ordered collections of schema definitions and axiomatic descriptions. There are complex scoping and naming rules, but the most important specification structuring mechanism is called schema inclusion. The name of a defined schema may be referred to in any other schema of axiomatic description after its definition, but entities within that schema may be referred to only if the schema is included in the signature of the following schema or axiomatic description.

There are operators for combining type compatible schemas (whose signatures don't disagree). There are also operators that support information hiding. The operator (\) hides members of a schema's signature as follows. Thus,

S\(x1,...,xn)

is the schema SNEW with x1,...,xn removed from S's signature and existentially quantified in S's property.

Schemas may also be parameterized. The collection of basic types known to the schema is extended to include the formal parameters. Generic schemas are instantiated by substituting the types of the actual parameters for the formal parameters.

Axiomatic descriptions can be used to define constants. Generic constants may also be defined using formal parameters (also called polymorphic constants), that is, a constant that represents a collection of constants with definite types.

Analysis of Z specifications usually means performing consistency and completeness checks, which validate the specification for accuracy and completeness, style, feasibility (sometimes called viability—seeing if a system state exists which satisfies the constraints specified in the initial condition), and expected properties. This analysis is performed by review by other specifiers who perform a "walk through" much like a code "walk through." Proofs are also used to analyze a Z specification, which is primarily done by hand, as there is no reliable automated prover for Z because a formal semantics does not yet exist.

Tools Applied

Several tools are available that provide support for checking for type and syntactic errors, as well as editors. The following is a brief description of these tools.

1. Editors

FORMALISER - a structure and text editor developed by Logica for PCs.

Genesis - a configurable formal methods support system built in LISP that supports Z among other formalisms. It consists of a specification structure editor and a proof editor.

LPEX - a "live parsing" syntax-directed editor developed for OS2 machines by IBM's Hursley Laboratory (see CICS case study in Volume 2 for details).

2. Type Checkers, Analyzers

FORSITE - a prototype tool developed in an Alvey project which consists of a WYSIWYG editor, a Z parser, a type checker, and an indexing facility for indexing schema and element names, a cross-referencing tool, a schema expansion facility, and a PostScript printing facility.

fUZZ - a tool that takes Z in ASCII form and processes it in LaTex to produce a formatted document. It checks conformance by type checking based on a standard Z library.

Zebra - an environment (toolsuite) for Z under development at Oxford University's Programming Research Group

Cadiz - another environment for Z developed by York University

SpecTra/Z - a hypertext tool developed at MCC's Software Technology Program

Roles Involved

A few specification writers and several reviewers who are able to read Z specifications.

Artifacts Produced

The primary artifact of Z is the single Z specification document, which is a sequence of schemas and natural language annotations describing the functional behavior of the system.

Metrics produced are the number of schemas, although it is also possible to measure schema complexity (number of schema variables and "depth" of schema nesting), as well as the number of refinement levels present in a specification.

The main representations are graphical, sometimes text, and in prototype, a hypertext representation developed at MCC in the SpecTra project.

B11.2 Observations

What the Method Achieves

Z is good at identifying errors that result from misconceptions in the model of a system. It is also good at determining and specifying relationships between levels of specification and design.

As seen in several of the cases that used Z, there is also the ability to reuse Z schemas, especially those that are generic.

Limitations

The lack of a standard Z - one that stipulates a method/framework/style conventions - could lead to collections of Z specifications that are incompatible (somewhat like the situation with VDM), although the development of a standard Z in the ZIP project is addressing this lack.

The lack of the requisite mathematical skills by requirements engineers to understand the underlying theory to Z is a bottleneck, as is the lack of available support tools (although efforts like Cadiz and Zebra are working on addressing this latter problem). Not much, however, is being done about the former problem.

Other Techniques Supported and Required

Abstraction and refinement skills are required to get the most out of Z.

Z supports designing through the use of constructing models, and although refinement techniques for Z are still the subject of research (e.g., the refinement calculus work by Carroll Morgan and others), Z does support a refinement approach to designing systems.

User Community

The main Z users are found in the U.K. and other European countries. Until recently, research and development on Z has been supported through ESPRIT and several U.K. government programs, although it is now somewhat self-supporting through commercial uses such as those described in some of the cases in this study. A fairly large number of U.K. and European universities teach Z courses (at last count approximately a year ago, the number was 35).

There are virtually no users—in terms of serious commercial organizations—of Z in North America other than Tektronix (who is not using Z currently). Z is being used as a specification language in an important upgrade to a clinical cyclotron control system at the Radiation Oncology Department of the University of Washington's Health Science Center. One government organization in the U.S. teaches Z, and a few North American universities offer Z courses.

B11.3 References

See [Dil90], [BSI91], [Gre91], [Spi88], and [Spi89].

REFERENCES FOR APPENDIX B

[A792] T. Alspaugh, S.R. Faulk, K.H. Britton, R.A. Parker, D.L. Parnas, and J.E. Shore, "Software Requirements for the A-7E Aircraft," NRL/FR/5530-92-9194, Aug. 1992.

[Abr91a] J.-R. Abrial, "Introduction to Set Notations," manuscript draft of July 1, 1991.

[Abr91b] J.-R. Abrial, "Abstract Machines," manuscript draft of July 1, 1991.

[Abr91c] J.-R. Abrial, "A Refinement Case Study," manuscript draft of January 9, 1991.

[Arc90] G. Archinoff, R.J. Hohendorf, A. Wassyng, B. Quigley, and M.R. Borsche, "Verification of the Shutdown System Software at the Darlington Nuclear Generating Station," International Conference on Control and Instrumentation in Nuclear Installations, Glasgow, May 1990.

[Bea91] S.P. Bear, "An Overview of HP-SL," VDM 91 Formal Software Development Methods, Springer-Verlag, Berlin, 1991, pp. 571-587.

[BSI91] British Standards Institute, "Z Standard," Preliminary Draft, May 1991.

[CD*92] M. Carnot, C. DaSilva, B. Dehbonei, and F. Meija, "Error-free Software Development for Critical Systems using the B Methodology," In Proceedings of the Third International Symposium on Software Reliability Engineering, IEEE Computer Society Press, 1992.

[deB80] J. de Bakker, *Mathematical Theory of Program Correctness* (Prentice-Hall, Englewood Cliffs, NJ, 1980).

[Dij76] E. Dijkstra, *A Discipline of Programming* (Prentice-Hall, New York, NY, 1976).

[Dil90] A. Diller, *Z: An Introduction to Formal Methods* (John Wiley & Sons, New York, NY, 1990).

[Dye92] M. Dyer, *The Cleanroom Approach to Quality Software Development* (John Wiley & Sons, New York, NY, 1992).

[EK85] S. Eckmann and R. Kemmerer, "INATEST: An Interactive System for Testing Formal Specifications," *ACM SIGSOFT Software Engineering Notes* 10(4), 17-18, 1985.

[Gre91] K. Greene, M. Bouler, D. Jamsek, S. Gerhard, and T. Ralston, *Z: Notation & Use, Formal Methods Transition Study, MCC, STP-FT-3 (revised), Final Report*, August 31, 1991.

[Gri81] D. Gries, *A Science of Programming* (Springer-Verlag, New York, NY, 1981).

[GVE90] R. Akers, B. Hartman, S. Smith, M. Taylor, and W. Young, "Gypsy Verification Environment User's Manual," Technical Report 61, Computational Logic Inc., September 1990.

[Har87] D. Harel, "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming* 8, 231-274, 1987.

[Hen78] K. Heninger, J. Kallendar, D. Parnas, and J. Shore, "Software Requirements for the A-7E Aircraft," NRL Memorandum Report 3876, November 1978.

[HP91] *Hewlett-Packard Journal*, December 1991, pp. 24-65, seven articles describing HP-SL and how it is used.

[Hoa69] C.A.R. Hoare, "An Axiomatic Basis for Computer Programming," *CACM* **12**(10), 1969.

[Hoa85] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall International, Series in Computer Science, (Prentice-Hall, New York, NY, 1985).

[Kem86] R.A. Kemmerer, D.R. Musser, D. Cooper, D.I. Good, K. Lovett, and D.H. Craigen, "Verification Assessment Study," Technical Report C3-CR01-86, National Computer Security Center, March 1986.

[Lev91] N. Leveson, M. Heimdahl, H. Hildreth, J. Reese, and R. Ortega, "Experiences using Statecharts for a System Requirements Specification," Proc. of the International Workshop on Software Specification and Design, Como, Italy, October 25-26, 1991, pp. 31-41.

[LS92] R. Linger and A. Spangler, "The IBM Cleanroom Software Engineering Technology Transfer Program," SEI Software Engineering Conference 1992, C. Sledge (editor), Lecture Notes in Computer Science 640, Springer Verlag, 1992.

[NHWG89] M. Neilsen, K. Havelund, K. Wagner, and C. George, "The RAISE Language, Method, and Tools," *Formal Aspects of Computing* 1, 85-114, 1989.

[Par90] D. Parnas, G.J.K. Asmis, and J.D. Kendall, "Reviewable Development of Safety Critical Software," International Conference on Control and Instrumentation in Nuclear Installations, Glasgow, May 1990.

[Ros84] A.W. Roscoe, "A Denotational Semantics for Occam," in S.D. Brookes, A.W. Roscoe, and G. Winskel, "Seminar on Concurrency, Carnegie-Mellon University, Pittsburgh, PA, July 9-11, 1984," Lecture Notes in Computer Science 197, Springer-Verlag, Berlin, pp. 306-329.

[Ros86] W.L. Roscoe, "The Laws of Occam," Oxford University Programming Research Group, Computing Science Laboratory, Technical Report, 1986.

[Sch80] J. Scheid, "Ina Jo: SDC's Formal Development Methodology," *ACM SIGSOFT Software Engineering Notes* 5(3), 1980.

[Sho67] J.R. Shoenfield, *Mathematical Logic* (Addison-Wesley, Reading, MA, 1967).

[Spi88] J. M. Spivey, *Understanding Z, A Specification Language and its Formal Semantics* (Cambridge University Press, New York, NY, 1988).

[Spi89] J.M. Spivey, *The Z Notation: A Reference Manual* (Prentice-Hall, New York, NY, 1989).

Appendix C

INITIAL QUESTIONNAIRE

This questionnaire was sent by the authors to formal methods study participants.

APPROACH

The study is being conducted by a team of experts in software engineering and formal methods. This questionnaire will be followed by structured interviews with participants in your project during which we will go into further detail on some of the questions. Soon thereafter, a summary of the data you provided and our interpretation will be returned for your comment. A report based on our survey will be published in the summer of 1992 and the results will be shared with you at the earliest possible time.

1. Organizational Context

Please describe, in general terms, the organization in which this project was conducted: its goals, products, structure, and composition at the time of the project.

2. Project Content and History

Describe the application involved. Please provide a time-line of the start, major milestones, and level of effort over the history of the project.

3. Application Goals

Why was this product developed and what were the major concerns that had an impact on the directions of the product development?

4. Formal Methods Factors

Why did you choose to use formal methods? Which formal method(s) did you choose? What criteria did you use in your selection process?

5. Formal Methods and Tools Usage

Describe how you used the method and what tool support you utilized.

6. Results

What advantages and disadvantages did you find using formal methods and tools on this application? What would you have done differently?

7. Additional Information

What additional information would help us understand your project before our visits? Are there materials that would be available to us before or after the visit?

Appendix D

QUESTIONNAIRE FOR STRUCTURED INTERVIEW

This second questionnaire was used by the authors to focus interviews.

D1. ORGANIZATIONAL CONTEXT

1. What are the general educational backgrounds of areas of expertise of the technical staff? What portion of the staff has had training in formal methods?

2. How does the organization handle the training of staff in new technologies? Is there in-house training? Do you go outside the organization to hire in the new areas?

3. Briefly describe the management structure.

4. Is there a company-wide system development process? If so, briefly describe it. How do you monitor the software development process?

5. What other information about the organization and the environment within which the project functioned may be of interest to this survey?

D2. PROJECT CONTENT AND HISTORY

1. What are the general educational backgrounds of areas of expertise of the technical staff? What portion of the staff has had training in formal methods?

2. How did the project handle the training of staff in new technologies? Was there in-house training? Did you go outside the organization to hire in the new areas?

3. Briefly describe the management structure.

4. Was there a project-wide system development process? If so, briefly describe it. How do you monitor the software development process?

D3. APPLICATION GOALS

1. How important was the application to the organization? How did the application area relate to the organization's business area? Were the goals of the development and the general goals of the organization in consonance with each other?

2. To what extent were the following of importance to you:

   - time-to-market?
   - economics? (e.g., reducing cost of development)
   - reliability of application?
   - quality of application?

3. What novelties were involved in developing the application? For example: Was this a new application domain? Were you using new technology?

D4. FORMAL METHODS FACTORS

The first questionnaire adequately addresses this area.

D5. FORMAL METHODS AND TOOLS USAGE

Formal methods process questions:

1. Did the decision to use formal methods modify the process in any manner? Where did formal methods fit into the process? What were the roles of the different techniques used in the process, e.g., use of testing? Was IV & V used?
2. Describe the part of the application which used the formal method(s). What other methods, if any, were used on other parts of the development?
3. Compared to other methods you have used, would you consider the formal method used on your project to be mature? Did the inclusion of the formal method impact the maturity of the process used in any manner?
4. Did you identify any risks from including formal methods in developing your application?
5. With respect to scalability, were you aware of previous developments using the formal method which successfully produced a system of roughly the same scale of complexity or size as the one in your project? If so, please indicate which type of system.
6. Would you consider the specification produced by the formal method (if any) more or less readable than other specifications produced by other methods you have used? Why? How would you describe the effect of the formal method on the reviewability of the specification by others (e.g., regulatory or certification bodies, if appropriate)?
7. What other information about the process may be of interest to this survey?

D6. FORMAL METHODS USAGE QUESTIONS

1. Describe the tool(s) used to support the formal methods and how, if at all, these tools were coordinated with other project tools. Was the presence of tools important in your choosing the formal method(s)?
2. Were the available tools supporting the formal method cost-effective, robust, easy to learn and to use? Any quantitative information to support the claims?
3. Did the formal method tool have support facilities for changes, such as browsing or cross-referencing? Briefly describe the features supporting such.
4. Was the interface for the tools associated with the formal method satisfactory (e.g., SDEs versus other types of editors, and in comparison to CASE tool interfaces)?
5. Were interfaces to other development tools necessary?
6. What other comments may be of interest to this study?

D7. RESULTS

1. What were the main results of the project?

2. What do you feel were the major consequences of using formal methods on your project?

3. Did the use of formal methods help you in reaching the application goals? If so, how? If not, why not?

4. Did the use of formal methods meet the criteria for choosing to use the technology?

5. What qualitative and quantitative results can you report for the project on the following factors? Can any conclusions be derived on the impact of formal methods on these results?

   - quality
   - reliability impact (including error discovery)
   - cost benefits
   - productivity of personnel
   - organizational confidence in application

6. For those errors and ambiguities that were uncovered, can you characterize how they were discovered? (For example, by formalizing requirements or by testing?)

7. Were error rates projected and compared to actuals? Do you have quantitative results?

8. If this was the organization's first experience with formal methods, how was the technology transitioned into the organization? Were any conclusions drawn on how to transition the technology in the future? Are formal methods still in use at the organization?

9. What would you do differently?

10. Were there any conclusions pertaining to training and education needed for using formal methods effectively? Any recommendations?

11. What other information do you feel would be of interest to this survey?

Appendix E

REVIEW COMMITTEE

The members of the Review Committee were chosen for their cumulative broad knowledge of Computing Science and interest in, and experiences with, surveying aspects of Computing Science. None of the members of the Review Committee could be viewed as particular proponents of formal methods. In fact, some could be viewed as being downright skeptical. The members of the committee were:

Lorraine Duvall
Duvall Computer Technologies, Inc.
Route 31, #2, Box 189
Canastota, New York 13032
U.S.A.

John Gannon
University of Maryland
Computer Science Department
College Park, Maryland 20742
U.S.A.

Morven Gentleman
Head, Software Engineering Laboratory
Institute for Information Technology
National Research Council
Ottawa, K1A 0R6
CANADA

Adele Goldberg
Parc Place Systems
999 E. Arquez Ave.
Sunnyvale, California 94086-4593
U.S.A.

John J. Marciniak
Director of Technologies
CTA Inc.
6116 Executive Blvd., Suite 800
Rockville, Maryland 20852
U.S.A.

The authors appreciated the input of the Review Committee, but our readers must note that the views taken in this report are those of the authors and may not reflect the views of the Review Committee.

Appendix F

ACKNOWLEDGMENTS

Support for this study was provided by organizations in Canada and the United States. The Atomic Energy Control Board of Canada provided support for Dan Craigen and for the typesetting and technical editing provided by Karen Summerskill. The U.S. Naval Research Laboratory, Washington, D.C., provided support for all three authors. The U.S. National Institute of Standards and Technology provided support for Ted Ralston.

The authors particularly wish to acknowledge the numerous individuals who agreed to participate in the study. Many of these individuals were sidetracked from their busy schedules, yet made the time available to fill out questionnaires and to be the subject of interviews. This study could not have been possible without their participation.

We also wish to thank our review committee, which consisted of Lorraine Duvall, John Gannon, Morven Gentleman, Adele Goldberg, and John Marciniak; and the reviewers from our sponsoring organizations, namely, Richard Taylor of AECB, Dolores Wallace and Rick Kuhn of NIST, and Connie Heitmeyer, Anne Rose, Alan Bull, Carolyn Gasarch, Bruce Labaw, Ralph Jeffords, Daniel Kiskis, and Paul Clements of NRL.

The authors wish to thank Karen Summerskill, ORA Canada, for her work on editing and typesetting this document.